VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Využití nástroje Weka v oblasti machine learning

Weka Tool in Machine Learning

2016 Michal Štěpán

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Michal Štěpán**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Využití nástroje Weka v oblasti machine learning
Weka Tool in Machine Learning

Jazyk vypracování: čeština

Zásady pro vypracování:

V rámci diplomové práce bude student pracovat se software Weka, který obsahuje sadu algoritmů strojového učení vhodných pro dolování dat. V rámci diplomové práce student prostuduje možnosti tohoto software z pohledu algoritmů, které jsou v něm implementovány, ale zejména se zaměří na koncepci tohoto software a postupů propojení jednotlivých modulů a toků dat. Dále se student zaměří na použití nástroje Weka pro zpracování dat velkých objemů, a to buď pomocí technologie Hadoop a nebo jeho nasazením na výpočetní nody HPC clusteru či systémy s velkou sdílenou pamětí. Přesný výběr technologie bude záviset na potřebách experimentů, které budou založeny na datech z oblasti detekce síťových útoků.

Jednotlivé body práce jsou:
1. Analýza a popis software Weka z pohledu algoritmů a propojení jednotlivých modulů.
2. Analýza použití Weka pro zpracování rozsáhlých dat.
3. Implementace experimentů se softwerem weka pro oblast detekce síťových útoků.
4. Vyhodnocení experimentů.

Seznam doporučené odborné literatury:

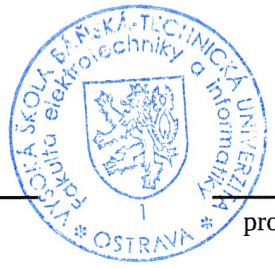
[1] Weka 3: Data Mining Software in Java, http://www.cs.waikato.ac.nz/~ml/weka/
[2] A. Abraham, R. Jain: Soft Computing Models for Network Intrusion Detection Systems, http://arxiv.org/pdf/cs/0405046v1.pdf

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Kateřina Slaninová, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016

Michal Tlač

.....

Rád bych na tomto místě upřímně poděkoval mé vedoucí práce Ing. Kateřině Slaninové, Ph.D. a Ing. Janu Martinovičovi, Ph.D. za možnost zpracovávat toto téma a také za jejich cenné podněty ke zlepšení a ochotný přístup.

Abstrakt

Tato práce se zabývá analytickým nástrojem Weka a jeho použitím ve strojovém učení. V jednotlivých kapitolách je popsáno prostředí nástroje spolu s algoritmy, kterými disponuje. Nástroj Weka je dále popisován z hlediska propojení jednotlivých modulů a možností paralelizace. V tomto nástroji jsme analyzovali datasety z problematiky detekce síťových útoků jak na osobním počítači, tak v prostředí HPC - přičemž jsme se zaměřili na možnosti nástroje a problematiku zpracování velkých datasetů. Pomocí nástroje Weka jsme úspěšně analyzovali datasety z oblasti detekce síťových útoků na výpočetních uzlech HPC a identifikovali jsme možnosti paralelizace. Na základě výsledků je možné jednotlivá zjištění dále zpracovávat.

Klíčová slova: detekce síťových útoků, HPC, modularizace nástroje Weka, paralelizace v nástroji Weka, strojové učení, Weka

Abstract

This thesis describes analytic tool Weka and its usage in machine learning. Following chapters describes tools' interface along with available algorithms. Weka tool is further analyzed from technical aspects like interconnection of modules and parallelization. We also analyzed datasets of intruder detection issue on both personal computer and HPC environment - we focused on possibilities of Weka tool and issue of processing large datasets. Intruder detection datasets were successfully analyzed on HPC nodes and parallelization possibilities were identified. Various findings could be further processed based on the results.

Key Words: HPC, intruder detection, machine learning, Weka, Weka tool modularization, Weka tool parallelization

Obsah

Seznam použitých zkratek a symbolů

AR	–	Asociation Rules
ARFF	–	Atribute-Relation File Format
CLI	–	Command Line Interface
DOS	–	Denial of service
DT	–	Decision Trees
GC	–	Garbage Collector
HPC	–	High Performance Computing
JNI	–	Java Native Interface
JVM	–	Java Virtual Machine
KFI	–	Knowledge Flow Interface
LGP	–	Linear Genetic Programming
MEP	–	Multi Expression Programming
MPI	–	Message Passing Interface
MP	–	Multilayer perceptron
NN	–	Neural Network
OOM	–	Out of memory
OP	–	Operační paměť
PBS	–	Portable Batch System
R2L	–	Remote-to-User
SVM	–	Support Vector Machines
U2R	–	User-to-Root

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Technologie moderního světa výpočetní techniky jdou stále kupředu. Chytrá elektronika ale i ostatní zařízení nám stále více ulehčují život. Technika se dostává tam, kde bychom ji před pár lety ani nečekali. S tím souvisí stále širší objem a sprektrum dat, které se k nám dostávají. Data jsou ale sama od sebe neužitečná - je z nich potřeba dostat informace. K tomu nám slouží technologie zvané strojové učení (machine learning) a data mining. Ten zažívá v posledních letech stále větší boom právě díky stále většímu objemu dat, která nás obklopují.

Cílem této práce je prostudovat možnosti anylytického nástroje Weka se zaměřením na koncepci tohoto nástroje a na postupy propojení jednotlivých modulů a toků dat. Dalším cílem je zaměřit se na použití nástroje Weka pro zpracování dat velkých objemů nasazením na výpočetní uzly HPC clusteru.

V úvodní části práce jsou uvedeny pojmy data mining a strojové učení a představeny základní skupiny algoritmů, které se v tomto odvětví používají. Z jednotlivých kategorií jsou představeny nejznámější algoritmy, které jsou dostupné i v nástroji Weka. V práci jsou uvedeny rovněž různé nástroje pro práci se strojovým učením. Převážná část práce se zabývá analytickým nástrojem Weka. Ve třetí kapitole jsou uvedeny jednotlivé možnosti používání nástroje Weka. Čtvrtá kapitola se zabývá problematikou detekce síťových útoků za použití nástroje Weka. V páté kapitole jsme se zaměřili na strukturu nástroje Weka z techničtějšího pohledu. V této kapitole jsme také analyzovali, jak je nástroj Weka konstruován a jak mezi sebou jednotlivé moduly komunikují. Také jsme analyzovali jednotlivé možnosti rozšíření a paralelizace. V neposlední řadě jsme se v této kapitole zaměřili na výhody a nevýhody plynoucí z použití platformy Java™. Šestá kapitola se zabývá zpracováním velkých datasetů v nástroji Weka. Analyzovali jsme zejména paměťovou náročnost zpracování velkých datasetů. Poslední kapitola se věnuje nasazení nástroje Weka na výpočetní uzly HPC. V této kapitole je prezentováno, jak nástroj Weka správně použít na výpočetnách uzlech HPC a zároveň jak analyzovat datasety z oblasti detekce průniků s využitím nabitých poznatků o možnostech nástroje Weka a jeho paralelizaci. Výstupem práce je také několik poznatků, jak efektivně a paralelně využít nástroj Weka na výpočetních uzlech HPC.

2 Strojové učení

2.1 Data vs. informace

Jak již bylo naznačeno v úvodu, dat máme nepřeberné množství. Ať už z počítačů, kde si ukládáme data na pevný disk, nebo internetu, kde se každá naše volba ukládá a odesílá, přes různé dotazníky, které vyplňujeme, až po čtení a ukládání dat z různých senzorů, které nás obklopují. Je ale nutné uvědomit si, že data samotná nám nic neřeknou. Nejdůležitější část je zjistit právě z těchto dat důležité informace, které ovlivní náš postup v budoucnosti.

Data jsou v této souvislosti chápany jako řetězce znaků, které představují formalizované vyjádření (zatím potencionální) informace, připravené pro přijetí lidmi nebo ke strojovému zpracování [1]. Data je zapotřebí zpracovat, komunikovat a vnímat, pak se teprve stávají informací.

Senzor na snímání počasí může zaznamenávat do databáze aktuální teplotu, vlhkost vzduchu, tlak a vítr. Nás ale ve skutečnosti zajímá, jestli si máme vzít kabát. Důležitá informace, která z těchto dat vyplývá, je, že venku je pocitově dobře a obejdeme se bez kabátu.

Právě zpracování dat do informací, neboli vyhledávání vzorců chování v datech, se zabývá analytická metodologie zvaná data mining.

2.2 Data mining

Data mining je relativně stará disciplína, jejíž počátky se datují někdy do 60. let 20. století, kde se vědci snažili využít znalostí regresní analýzy a rozhodovacích stromů k získávání informací z dat. Ale až s postupným zrychlováním počítačů a rychlým nárustem paměti spolu s rozšiřováním poznatků ze statistiky se tato metodologie systematicky začala používat v praxi. Dnes je obvyklé, že se tato metodologie používá jak na akademické půdě, tak i v komerční sféře.

Definice data miningu je následující [2]: Jedná se proces objevování vzorů chování v datech. Takový proces musí být automatický nebo (častěji) poloautomatický. Objevené vzory chování musí mít význam v tom smyslu, že z nich plyne nějaká výhoda (nejčastěji ekonomická).

2.3 Strojové učení

Strojové učení je podoblast umělé inteligence, která se zabývá algoritmy a technikami, které umožňují počítačovému systému "učit se"[3].

Zvažme příklad rozpoznávání čísel z rukopisu. Máme k dispozici obrázky velikosti 28 x 28 pixelů, což můžeme reprezentovat jako vektor x o velikosti 784 reálných hodnot. Cílem je získat správnou číslici 0 až 9 ze vstupního vektoru x. Jedná se o netriviální úkol, jelikož variace všech možných rukopisů je obrovská.

Díky přístupu strojového učení můžeme získat kvalitní rozpoznávací algoritmus. Vstupními daty, která označíme jako *tréninková*, vyladíme parametry adaptivního modelu. Výstupní hodnotu takových dat většinou známe (v tomhle případě musí tréninkový set rukopisů někdo ručně

oštítkovat). Výstupní hodnotu si představme jako cílový vektor *t*. Pro každý obrázek rukopisu existuje právě jeden cílový vektor *t*.

Algoritmus strojového učení můžeme vyjádřit jako funkci *y(x)*, která vstupnímu obrázku x přiřadí výstupní vektor y stejným způsobem jako cílové vektory. Přesná podoba funkce *y(x)* je rozhodnuta právě během tréninkové fáze, (nebo-li *učící*) na základě tréninkových dat. Jakmile je takový model vytvořen a naučen, může pak rozpoznávat hodnoty nových rukopisů, které označíme jako *testový* set. Samotný proces rozpoznávání obrázků, které se liší od těch, použitých v tréninkovém setu, se nazývá proces *generalizace*. V praktickém světě pak tréninkový set obsahuje pouhý zlomek všech různých možných vstupních vektorů. Proto je generalizace hlavním cílem v oblasti hledání vzorů.

Ve většině praktických úloh jsou pak vstupní data *předzpracována* tak, že výsledný problém je jednodušší vyřešit. V našem případě mluvíme o předzpracování dat takovém, že všechny vstupní obrázky byly před učením škálovány do stejné velikosti. Takový úkon nám pak značně zjednoduší rozpoznávání. Nutno podotknout, že pokud budeme předzpracovávat data, tak musíme předzpracovávat stejně data tréninková i testová, jinak bude učení nespolehlivé.

Z hlediska učení máme několik kategorií [4]:

- **učení s učitelem** - pro vstupní vektory známe cílové vektory. Příklad s rozpoznáním rukopisu, kde je naším cílem zařadit každý vstupní vektor k jedné konkrétní hodnotě, je příkladem úkolu *klasifikace*. Pokud je výstupem jedna nebo více spojitých hodnot, jedná se o úkol *regrese*.

- **učení bez učitele** - pro vstupní vektory neznáme cílové vektory. Cílem je najít v datech shluky podobných dat (technika *shlukování*), nebo rozhodnout o rozdělení dat ve vstupním prostoru (technika *odhadování hustoty*), nebo zobrazit data z mnohadimenzového prostoru do dvou nebo tří dimenzí (technika *vizualizace*).

- **posílené učení** (reinforced learning) - problém hledání vhodných akcí v dané situaci pro maximalizaci zisku. Zde není algoritmu znám očekávaný výstup, ale musí se sám rozhodnout metodou pokus a omyl. Většinou se jedná o sekvenci akcí, při které učící algoritmus komunikuje s prostředím. Příkladem je pak hra šachů počítače proti člověku.

2.4 Algoritmy

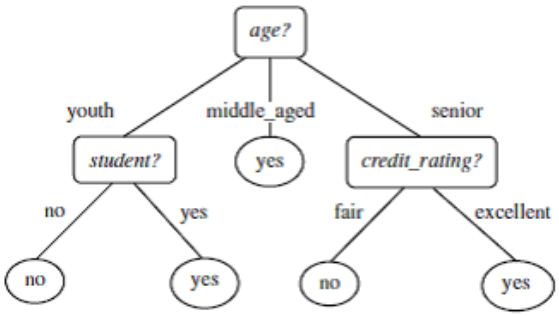
V oblasti strojového učení se setkáváme s velkým množstvím různých skupin algoritmů. Každá skupina je vhodnější na jiný typ úloh (jiný algoritmus použijeme na klasifikační problém, další pak zase na shlukovací problém, kde vstupnímu vektoru mohou chybět hodnoty). Tato kapitola stručně představuje základní skupiny algoritmů, se kterými se setkáváme ve strojovém učení a v nástroji Weka. U každé skupiny je představen nejznámější algoritmus, který je zároveň dostupný v nástroji Weka.

2.4.1 Rozhodovací stromy

Rozhodovací strom je nástroj, který přiděluje pravděpodobnost ke každé možné volbě na základě kontextu rozhodnutí: $P(f|h)$, kde f je prvek budoucího slovníku (množina možností) a h je historie (kontext rozhodnutí) [5]. Tato pravděpodobnost $P(f|h)$ je odvozená ze sekvence otázek $q_1, q_2, ...q_n$ ohledně kontextu, kde i-tá otázka je jednoznačně odvozená z odpovědí $i-1$ předešlých otázek.

Rozhodovací strom je reprezentován binárním stromem. Prostorová složitost vytvoření takového stromu je poměrně malá; ještě menší složitost je ale během jeho používání - *O(log N)*, kde N je počet instancí. Toto je velice důležité, protože učení tréninkového setu by mělo být rychlé. Další výhodou je, že stromy jsou relativně lehké na pochopení a používání. Proto je tato technika v oblasti strojového učení často používaná.

Rozhodování v takovém rozhodovacím stromu pak začíná u kořene, kde se postupně ptáme na vlastnosti vektoru a postupujeme dolů stromem, dokud nenarazíme na list - v takovém případě jsme získali klasifikaci daného vstupního vektoru. Ukázku takového rozhodovacího stromu vidíme na obrázku 1.



Obrázek 1: Příklad rozhodovacího stromu [6]

2.4.1.1 ID3 Algoritmus ID3 je jeden z algoritmů skupiny rozhodovacích stromů. U rozhodovacích stromů stanovuje pořadí rozhodovacích kritérií v každém kroku metrika. Existuje několik druhů metrik - ta nejzákladnější je **informační zisk**, který rozhoduje o pořadí kritérií v algoritmu ID3. Informační zisk lze definovat následovně [7]:

$$Gain(S, F) = Entropy(S) - \sum_{f \in values(F)} \frac{|S_f|}{|S|} Entropy(S_f) \tag{1}$$

kde S je set vstupních vektorů, F je jedna z možných vlastností vstupního vektoru a $|S_f|$ je počet vstupních vektorů setu S, které mají hodnotu f pro vlastnost F. Entropii získáme následovně:

$$H(p) = -\sum_i p_i * log2 * p_i \tag{2}$$

Při rozhodování, kterou vlastnost vektoru použijeme na dané úrovni stromu, pak porovnáme infomační zisk všech vlastností vstupního vektoru a ten, který má největší informační zisk, se stane pravidlem. Dané pravidlo pak z vlastností vektoru odstraníme a rekurzivně pokračujeme dále stromem. Tento postup je základ pro algoritmus ID3. V nástroji Weka máme k dispozici algoritmus J48 - ten sice vychází z algoritmu C4.5, ten je ale mírně vylepšenou verzí algoritmu ID3 (dokáže pracovat se spojitými i diskrétními hodnotami, používá *prořezávání* (angl. *pruning*) pro řešení problému *přeučení* (angl. *overfitting*).

2.4.1.2 Ovlivňující faktory Při tvorbě rozhodovacích stromů má vliv na výslednou kvalitu učení několik faktorů [8]:

- **přeučení** - jinými slovy opak generalizace. Výsledná pravidla fungují správně pro daný úkol, nicméně pravidla jsou moc úzce spjatá s daty a tak se nehodí na ostatní úkoly. Problém s přeučením můžeme řešit několika způsoby. Tím jednodušším je omezit velikost stromu. Tím ale můžeme ztratit důležité vlastnosti a může se nám zvýšit chybovost a tedy kvalita modelu. Dalším řešením je prořezávání.
- **prořezávání** - jedná se o techniku zmenšování objemu stromu a zjednodušení pravidel. Existují dva druhy prořezávání - *předprořezání* (angl. *pre-pruning*) a *poprořezání* (angl. *post-pruning*). Prořezávání má tu výhodu, že nezvyšuje chybovost modelu.
- **spojité proměnné** - výše uvedené příklady ilustrují správné použití rozhodovacích stromů při klasifikaci diskrétní proměnné. Problém ale nastává při proměnné spojité - je zapotřebí uvažovat o všech hodnotách jako diskrétních? Nebo vytvořit intervaly? A kolik? Spojité proměnné nám dělají výsledný model složitější.
- **chybějící hodnoty** - s chybějícími hodnotami se rozhodovací stromy vyrovnávají dobře - místo chybějící hodnoty zvolí medián z možností dané vlastnosti.
- **rozmanitost** - jedná se o vlastnost, která výrazně ovlivňuje výslednou velikost stromu. Pokud je rozmanitost v datech velká, často vzniká problém přeučení.

2.4.1.3 Shrnutí Použití rozhodovacích stromů je při minimalizaci negativních faktorů dobrý způsob, jak rychle a efektivně klasifikovat data. Abraham ve svém experimentu [9] porovnával výsledky a rozhodovacích stromů a přišel na to, že rozhodovací stromy jsou přesnější s menšími tréninkovými sety dat.

2.4.2 Asociační pravidla

Asociační pravidla jsou v strojovém učení oblíbená kvůli své jednoduchosti. Můžeme si je představit jako sérii *if - else* příkazů nad vlastnostmi vstupních dat, které nám úspěšně klasifikují data. Taková pravidla můžeme jednoduše vyčíst i z rozhodovacích stromů - takto vyčtená pravidla budou jednoznačná a nebude záviset na jejich pořadí. Problémem je ale jejich přehnaná

složitost, kvůli které se musí sada pravidel prořezat, abychom získali minimální sadu pravidel. Tato operace ale vyžaduje další výpočetní výkon. Tato metoda generování pravidel z rozhodovacích stromů je tedy velice pomalá.

Při vytváření pravidel je pak nejčastěji používaná technika *pokrytí* - základní myšlenka je zahrnout co nejvíce instancí požadované třídy a zároveň vynechat co nejvíce instancí ostatních tříd. Předpokládejme, že nové pravidlo zahrnuje t instancí, kde p jsou kladné cílové vektory a t - p jsou ostatní cílové vektory. Tyto ostatní cílové vektory označují chybovost vybraného pravidla. Úkolem je vybrat takové pravidlo, které bude mít maximální poměr p/t.

2.4.2.1 PART *PART* je známý algoritmus ze skupiny rozhodovacích pravidel, který můžeme najít v nástroji Weka. Jeho základem je použití techniky *separate-and-conquer*, definována takto [10]: Nauč se pravidlo, které pokrývá část tréninkového datasetu (část *separate*) a nauč se rekurzivně další pravidlo, které pokrývá další se zbývajících instancí (část *conquer*, dokud nezůstanou žádné instance. Algoritmus *PART* vytvoří v každé iteraci částečný C4.5 rozhodovací strom a nejvhodnější list použije jako pravidlo. Algoritmus si dobře poradí se spojitými, diskrétními i chybějícími hodnotami.

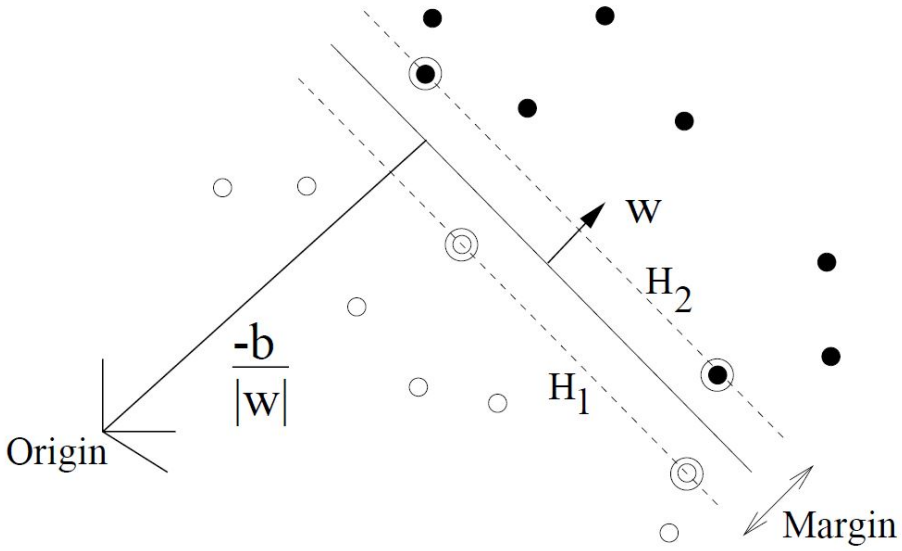
2.4.2.2 Shrnutí Asociační pravidla jsou velice podobná rozhodovacím stromům. Možná i díky tomu se kvůli lepší přehlednosti a kvalitám rozhodovacích stromů dává přednost právě stromům. Nicméně i asociační pravidla si své místo v řadě praktických úloh najdou.

2.4.3 Support Vector Machines

Algoritmus SVM je oproti metodě asociačních pravidel a rozhodovacích stromů značně složitější již při její reprezentaci. Nicméně díky své linearitě je v poslední době na vzestupu a nahrazuje tak jiné metody strojového učení. Koncept SVM přehledně popsal ve své knize Marsland [7], případně Zaki [11].

Cílem SVM je najít nadrovinu, která v prostoru instancí optimálně rozdělí body tak, že leží v opačných poloprostorech a hodnota minima vzdáleností bodů od roviny je co největší. Na popis takové nadroviny stačí pouze nejbližší body, které nazýváme *podpůrné vektory* (angl. *support vector*). Tato metoda je binární - rozděluje data pouze do dvou tříd. Rozdělující nadrovina je lineární funkcí v prostoru instancí.

Obrovskou výhodou SVM oproti stromům a pravidlům je ta, že umí klasifikovat nejen objekty v $\mathbb{R}^n$, ale i například stromy, grafy a sekvence. Hlavním principem je tzv. *jádrová transformace* (angl. *kernel transformation*), která prostor, ve kterém se vyskytují instance, převede do prostoru vyšší dimenze. Tak se stane původně lineárně neseparovatelná úloha lineárně separovatelnou, na kterou lze již aplikovat algoritmus, který najde rozdělující nadrovinu. Ilustraci takové nadroviny vidíme na obrázku 2. Z počátku je vyveden vektor $\frac{-b}{|w|}$, na nějž nacházíme kolmou nadrovinu. Ta se hledá tak, aby mezera M byla co největší. K tomu použijeme nejbližší body jako podpůrné vektory. [12]



Obrázek 2: Hledání nadroviny v prostoru instancí algoritmu SVM [12]

2.4.3.1 Shrnutí I když se jedná o poměrně mladou techniku, SVM je hojně používaná v oblasti strojového učení, právě díky řešení lineárně neseparovatelných úloh. Oproti rozhodovacím stromům ovšem disponuje větším množství parametrů, tudíž může být pro méně zkušeného člověka těžší tuto techniku správně používat.

V základní verzi nástroje Weka se algoritmus SVM neobjevuje, ovšem lze jej doinstalovat.

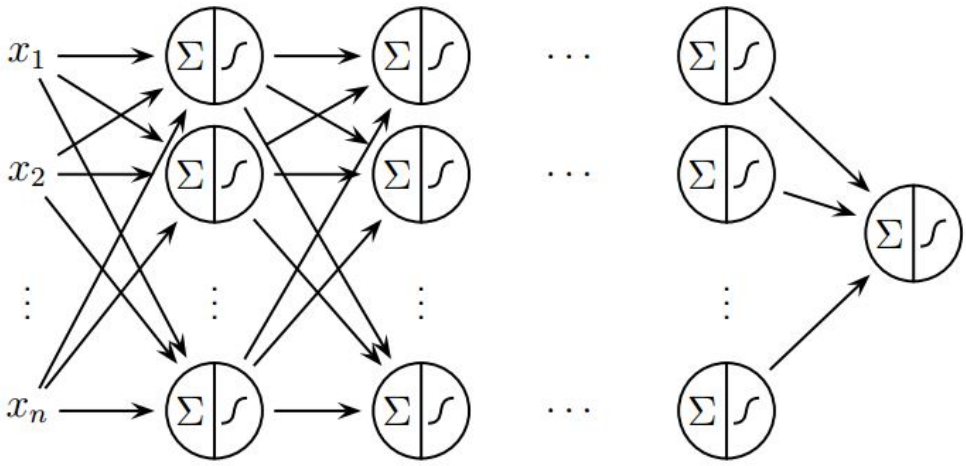
2.4.4 Neuronové sítě

Neuronové sítě jsou inspirovány biologickými neuronovými sítěmi a využívají distribuované, paralelní zpracování informace při provádění výpočtů [13]. Paměť a zpracování informace v neuronové síti je spíše globální, než lokální. Znalosti jsou ukládány především prostřednictvím síly vazeb mezi jednotlivými neurony. Učení je základní a podstatná vlastnost neuronových sítí. Jednotlivé neurony ovšem nejsou schopny vyřešit složité úlohy a proto vytváříme sítě pomocí známých algoritmů.

2.4.4.1 Multilayer perceptron Základním algoritmem pro vytváření neurononových sítí je *multilayer perceptron*. Jedná se o konečný acyklický graf, kde uzly jsou neurony s přenosovou funkcí. Taková síť má několik vrstev neuronů (obrázek 3):

- vrstva vstupních neuronů - uzly, které nejsou cílem žádného spojení (nemají žádný vstup). Obyčejně je počet neuronů na této vrstvě shodný s počtem vstupů.

- vrstva výstupních neuronů - uzly, které nejsou zdrojem žádného spojení (nemají výstup). Počet neuronů na této vrstvě záleží na tom, jak jsou popsány cíle úlohy (jaké jsou požadované výstupy).
- skrytá vrstva - uzly, které nepatří do vstupní ani výstupní vrstvy.



Obrázek 3: Znázornění vrstev neuronů v neuronových sítích [14]

Jelikož může být ve skryté vrstvě obrovské množství neuronů (záleží na druhu a povaze úlohy), je vytváření takovéto sítě výpočetně i paměťově náročné. Tato síť je tzv. *forward propagated* - tzn. že neurony na i-té vrstvě mohou mít spojení pouze na neurony na $i+n$-té vrstvě.

Vlastní trénování neuronu probíhá většinou iterativně, kdy algoritmus jednotlivé prvky tréninkového datasetu předkládá postupně neuronu, zjišťuje jeho odezvu na předložený vstup a na základě odchylky jeho výstupu od výstupu požadovaného provádí korekci vah neuronu. Interval, ve kterém dojde k předložení všech vzorů tréninkové množiny alespoň jednou, nazýváme epochou učení. K naučení sítě může být dle komplexnosti problému zapotřebí desítky až tisíce epoch [15].

2.4.4.2 Shrnutí Neuronové sítě patří mezi velice kvalitní algoritmy. Nevýhodou ovšem je velká náročnost vytváření sítě. Abychom dosáhli kvalitních výsledků, zpravidla potřebujeme větší množství epoch - to se pak odráží na celkové době učení, která je mnohem větší, než u ostatních skupin algoritmů.

2.4.5 Shlukování

Předchozí uvedené algoritmy byly příkladem algoritmů pro klasifikaci dat. Shlukování funguje na bázi učení bez učitele - tudíž cílem je najít takové shluky, které by nejlépe popisovaly vstupní data. Tento princip se používá převážně pro seskupování dat do skupin, ovšem je možné jej použít i pro klasifikační úlohy a to tak, že vynecháme třídní parametr. Některé algoritmy shlukování pak dovolují, aby jedna instance vstupu patřila do více shluků.

2.4.5.1 k-means Známým algoritmem je *k-means*, u kterého specifikujeme parametr k - požadovaný počet shluků. k bodů je potom náhodně vybráno jako středy shluků a na základě zvolené míry jsou instance přiřazeny k nejbližšímu shluku. Následně se vypočtou průměry vzdáleností v každém shluku a ty pak slouží jako nové středy. Následně se znova dle zvolené míry zařadí všechny instance do shluků a celý proces se opakuje do té doby, než jsou shluky stabilizovány a neměnné. Základní instalace nástroje Weka nabízí několik druhů míry - Euclidean, Chebyshev, Filtered, Manhattan a Minkowski.

2.4.5.2 Shrnutí Algoritmy shlukování mají díky svému rozdílnému využití oproti klasifikačním algoritmům své uplatnění v různých problémech. V nástroji Weka můžeme využít tyto algoritmy při učení s učitelem i při učení bez učitele.

2.4.6 Ostatní

Mezi další skupiny algoritmů, které jsou ve strojovém učení využívány a zároveň jsou dostupné v nástroji Weka, jsou například bayesovské algoritmy, jež jsou založené na naivní bayesovské klasifikaci. Alternativně se tomuto přístupu říká pravděpodobnostní přístup, protože je založen čistě na statistických metodách a pravděpodobnosti. V nástroji Weka máme rovněž k dispozici i různé statistické metody, jako je například *lineární regrese*.

2.5 Nástroje pro práci s ML

Existuje velké množství nástrojů pro práci se strojovým učením. Nástroje jsou psány v různých programovacích jazycích - Python, Java, C++, C#, JavaScript, aj. V této kapitole přiblížíme nejznámější nástroje pro strojové učení.

2.5.1 Mahout

Framework Mahout [16] byl dlouho spjat s technologii Hadoop, nicméně mnoho algoritmů lze spustit i mimo tuto technologii. Tímto se stávají užitečnými pro samostatné aplikace, které mohou být eventuálně převedeny pod Hadoop nebo naopak pro Hadoop projekty, které mohou být převedeny do samostatných aplikací. Avšak nevýhodou tohoto frameworku je, že jen malé množství jeho algoritmů podporuje nový výkonný Spark framework pro Hadoop a používají nyní již zastaralý MapReduce framwork.

2.5.2 CUDA-Convnet

CUDA-Convnet [17] těží, jak už název vypovídá, z vysoké výkonnosti jader grafický karet, které řeší určité problémy mnohem rychleji, než běžný CPU. Tato aplikace je napsána v C++ a slouží k použití s algoritmy neuronových sítí. Nevýhoda této aplikace je nutnost grafické karty NVidia nové generace (alespoň Fermi) a zároveň malá výkonnost na výpočetních uzlech HPC.

2.5.3 Azure Machine Learning Studio

Jedná se o kolaborativní drag-and-drop nástroj [18], který slouží k vytváření, testování a nasazení prediktivních řešení na data. Azure Machine Learning Studio je cloudové řešení, které publikuje vypočtené modely skrze webové služby, které mohou být jednoduše využity v různých aplikacích nebo bussiness inteligence nástrojích (např. Excel). Tento nástroj využívá jak algoritmy psané v jazycích Python a R, tak své vlastní Microsoft algoritmy.

2.5.4 Weka

Analytický nástroj Weka [19] je psaný v Javě a jeho výhoda je ve velkém množství operací, které lze nad daty provádět - preprocessing, filtrace dat, shlukování a klasifikace, hledání atributů aj. Zároveň nabízí několik možností spouštění pro širší využití na různé problémy. Relativní nevýhodou nástroje Weka může být načítání datasetu do OP počítače - tento problém se dá ale řešit instalací volitelných balíčků (přispívaných komunitou), případně pokročilejším používáním nástroje.

3 Analytický nástroj Weka

Počátek se datuje do roku 1993, kde v Univerzitě Waikato na Novém Zélandě začali vývoj původní verze nástroje Weka [19]. V roce 1997 ovšem padlo rozhodnutí předělat nástroj Weka od základu v Javě, což dalo základ nástroje tak, jak jej známe dnes. Nástroj se stále zdokonaluje a v době psaní této diplomové práce se nachází ve verzi 3.7.13, uvolněné v září 2015. Nástroj je volně dostupný pod GNU licencí.

Weka je nástroj, který obsahuje kolekci vizualizačních nástrojů a algoritmů pro datovou analýzu a prediktivní modelování s grafickým rozhraním pro snadný přístup k těmto funkcím. Weka podporuje několik standardních data miningových úloh, konkrétně preprocessing dat, shlukování, klasifikaci, regresi, vizualizaci a analýzu příznaků. Omezením je absence algoritmů a metod pro zpracování sekvencí.

3.1 Rozšířitelnost

Nástroj Weka má širokou základnu aktivních uživatelů, kteří do nástroje přispívají svými algoritmy a řešeními v podobě balíčků, které lze jednoduše do nástroje importovat. Tyto balíčky umožňují přidat do nástroje Weka implementace algoritmů, které v základní verzi nejsou. Nástroj Weka spravuje balíčky v *package manageru*, který je popsán v kapitole 3.5.5.

Mezi užitečné balíčky patří například *distributedWekaHadoop* [20], který umožňuje jistou míru spolupráce mezi nástrojem Weka a technologií Hadoop. Praktické využití tohoto balíčku je nad rámec této diplomové práce.

Mezi další užitečné balíčky patří *WekaGP 1.1*[21], který implementuje do nástroje Weka genetické programování. Dalším zajímavým balíčkem je *weka-spectral-clusterer* [22], který implementuje do nástroje Weka možnosti spektrálního shlukování.

Zajímavým projektem je také *Weka-Parallel* [23], což je modifikace nástroje Weka se záměrem maximálního paralelního zpracování úlohy tak, aby se zvýšila celková výkonnost nástroje. Detailněji se zabýváme projektem Weka-Parallel v kapitole 5.5.2.1.

Weka se takto stává silným nástrojem na poli strojového učení, převážně díky své rozšiřitelnosti.

3.2 Zpracovatelná data

Nástroj Weka umí zpracovávat pouze data textová a to za předpokladu, že se data nachází v jediném souboru nebo relaci, kde každý záznam obsahuje fixní počet atributů. Tento přístup do jisté míry omezuje použití nástroje.

Nástroj Weka si poradí s většinou běžných textových formátů včetně formátu CSV. Speciální formát nástroje Weka je formát ARFF. Jedná se o formát dat, který obsahuje výčet atributů spolu s anotacemi oddělující jednotlivé části. Příklad vstupních dat ve formátu ARFF můžeme vidět ve výpisu 1.

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast ,83,86, FALSE,yes
rainy ,70,96, FALSE,yes
rainy ,68,80, FALSE,yes
rainy ,65,70, TRUE,no
overcast ,64,65, TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
rainy ,75,80, FALSE,yes
sunny,75,70,TRUE,yes
overcast ,72,90, TRUE,yes
overcast ,81,75, FALSE,yes
rainy ,71,91, TRUE,no
```

Výpis 1: Vstupní data ve formátu ARFF

Nástroj Weka je schopen načítat data z pevného uložiště, z relační databáze na vzdáleném serveru a z URL webové stránky, přičemž platí stejné pravidlo, že data musí být textová.

Požadavek na textový formát vstupních dat je striktní pro základní instalaci nástroje Weka. Pokud potřebujeme analyzovat jiná data, než textová, můžeme se pokusit takový přístup implementovat.

3.3 Algoritmy

Algoritmy, které nástroj Weka obsahuje, lze rozdělit do několika základních skupin.

Pro klasifikační úlohy nabízí základní verze nástroje Weka několik skupin algoritmů:

- **bayesovské algoritmy** - jedná se o algoritmy, založené na principu pravděpodobnosti.
- **rozhodovací stromy** - skupina algoritmů, jejichž princip byl popsán v kapitole 2.4.1
- **asociační pravidla** - skupina algoritmů, jejichž princip byl popsán v kapitole 2.4.2
- **funkce** - zde se objevují algoritmy pracující na bázi statistických metod jako je například *lineární regrese*. Také se zde objevuje metoda *SVM*, jež byla popsána v kapitole 2.4.3. Je zde rovněž algoritmus *MultilayerPerceptron*, který je příkladem *neuronových sítí*, zmíněných v kapitole 2.4.4
- **lazy** - tyto algoritmy jsou založeny na principu klasifikace podle nejbližších instancí, nikoliv podle celého datasetu.
- **meta** - ostatní algoritmy pro práci s celým datasetem. Zde můžeme najít například algoritmy **iterativního učení** nebo metody jako *Stacking*, *Bagging* pro kombinaci více algoritmů

Shlukovací úlohy pak obsahují různé algoritmy shlukování. Proces shlukování je popsán v kapitole 2.4.5.

Tento výčet algoritmů je konečný pouze pro základní instalaci nástroje Weka. Do nástroje Weka lze implementovat další vlastní algoritmy. Příkladem může být implementace *genetického programování* nebo *spektrálního shlukování*, zmíněných v kapitole 3.1.

3.4 Evaluace výsledků

Nástroj Weka umožňuje několik způsobů, jak evaluovat výsledky učení modelu:

- Evaluaci na tréninkovém datasetu
- Evaluaci na dodaném testovém datasetu
- Procentuální rozdělení tréninkového datasetu
- Křížová validace (angl. cross-validation)

V následujících kapitolách jsou popsány všechny možnosti [24].

3.4.1 Evaluace na tréninkovém datasetu

Pokud zvolíme tuto možnost evaluace, naučený model bude testován nad datasetem, nad kterým se model učil. Nejedná se o spolehlivou metodu evaluace, jelikož je výsledek úzce spjatý s testovacím datasetem a tudíž nové instance obsahující jiné hodnoty atributů nemusí být správně klasifikovány.

Pokud ovšem požadujeme rychlou evaluaci, zda-li je náš zvolený algoritmus vhodný na daný typ úlohy, může tato metoda posloužit jako vhodná a rychlá kontrola.

3.4.2 Evaluace na dodaném testovém datasetu

Tato metoda evaluace se blíží reálnému použití, kde máme k dispozici dva datasety:

- Tréninkový dataset - dataset, nad kterým budujeme učící model
- Testový dataset - dataset, nad kterým aplikujeme učení

Pokud tedy splňujeme tuto podmínku, tato metoda evaluace je pro nás nejvhodnější, jelikož nám umožňuje naučený model testovat na reálných datech.

Nástroj Weka klade na tuto metodu omezující faktor - tréninkové i testové datasety musí být totožné z hlediska počtu atributů a pořadí atributů. Může zde nastat typický problém: tréninkový dataset obsahuje výstupní vektor, testový dataset nikoliv. V tomto případě Weka bez explicitní úpravy nedokáže evaluaci provést.

Výhodou této metody je jako v předchozí metodě rychlá evaluace.

3.4.3 Procentuální rozdělení tréninkového datasetu

Procentuální rozdělení vstupního datasetu použijeme tehdy, kdy nemáme k dispozici testový dataset, avšak nechceme testovat naučený model na stejných datech. V tomto případě předložíme nástroji Weka parametr n, který značí procentuální rozdělení vstupního datasetu. Pokud například zvolíme číslo 66, vstupní dataset bude rozdělen na 66% tréninkovou část a 34% testovou část. Způsob evaluace je pak stejný, jako v předchozí metodě.

Z hlediska rozdělení vstupních dat evidujeme dva přístupy:

- s náhodným rozdělením - instance vstupního datasetu jsou zamíchány, rozdělení instancí mezi tréninkový a testový dataset je tedy náhodné
- bez náhodného rozdělení - rozdělení instancí mezi tréninkový a testový dataset záleží na pořadí, ve kterém se vyskytují ve vstupním datasetu

Opět se jedná o poměrně rychlou metodu, která dokáže lépe reflektovat změny hodnot atributů v nových instancích, než metoda evaluace na tréninkovém datasetu.

3.4.4 Křížová validace

Jedná se o statisticky nejspolehlivější metodu evaluace výsledků, která nachazí využití v širokém spektru problémů strojového učení.

Uvažujme dataset s m instancemi a k atributy, znázorněnými v matici A $m \times k$. Nejpoužívanější technikou je n-skupinová křížová validace (angl. n-fold cross-validation) [23], kde parametr n udává, na kolik skupin se má vstupní dataset rozdělit pro evaluaci. Matice A je horizontálně rozdělena do n skupin. Pro každou z n iterací, je daná skupina vybrána jako *testový dataset* a zbylé skupiny jsou použité jako *tréninkový dataset*. Zvolený algoritmus strojového učení na takto vytvořeném tréninkovém datasetu naučí model a posléze na testovém modelu tento model ověří. Celková přesnost je vypočtená průměrem úspěšností klasifikací všech n skupin. Nejčastěji používanou variantou je rozdělení do 10 skupin. Tato metoda rovněž zajišťuje náhodné rozdělení instancí před rozdělením do skupin.

Tato metoda je velice přesná, ovšem časově náročná. Oproti výše uvedeným metodám je tato metoda průměrně n-krát pomalejší, n uvádí počet skupin, na které vstupní dataset rozdělujeme.

Časová náročnost křížové validace je jeden z důvodů, díky kterému vznikl balíček *Weka-Parallel*.

3.5 Popis jednotlivých částí

Podrobnému popisu uživatelského rozhraní nástroje Weka se již věnoval Witten a Frank [2]. Cílem této sekce je přiblížit základní postupy a orientaci v uživatelském rozhraní nástroje Weka.

Po spuštění nástroje nás uvítá úvodní obrazovka s možností výběru požadované volby. Postupně budou představeny všechny možnosti dle obrázku 4.

Obrázek 4: Uvítací obrazovka po spuštění nástroje Weka

3.5.1 Explorer

Explorer je hlavní grafické rozhraní nástroje Weka, obsahující 6 různých panelů, korespondujících s různými typy úkolů v oblasti data miningu.

Panel *Preprocess* slouží k předzpracování dat pro následné učení. Tento panel umožňuje načítat dataset z různých zdrojů (dle kapitoly 3.2) a stejně tak dokáže data generovat a předzpracované datasety také ukládat. K předzpracování slouží položka filtr, díky které můžeme na data aplikovat různé filtry tak, abychom zajistili vhodnost modelu pro učící úlohy. Rovněž tento panel nabízí vizualizaci vstupních dat v podobě grafů. Panel *preprocess* můžeme vidět na obrázku 5.

Panel *Classify* slouží k vytváření modelů klasifikačních úloh. Po zvolení algoritmu, způsobu evaluace výsledku a dalších volitelných možnostech nám nástroj Weka poskytne informace o výsledku naučeného modelu. Takový model také můžeme v závislosti na zvoleném algoritmu vizualizovat.

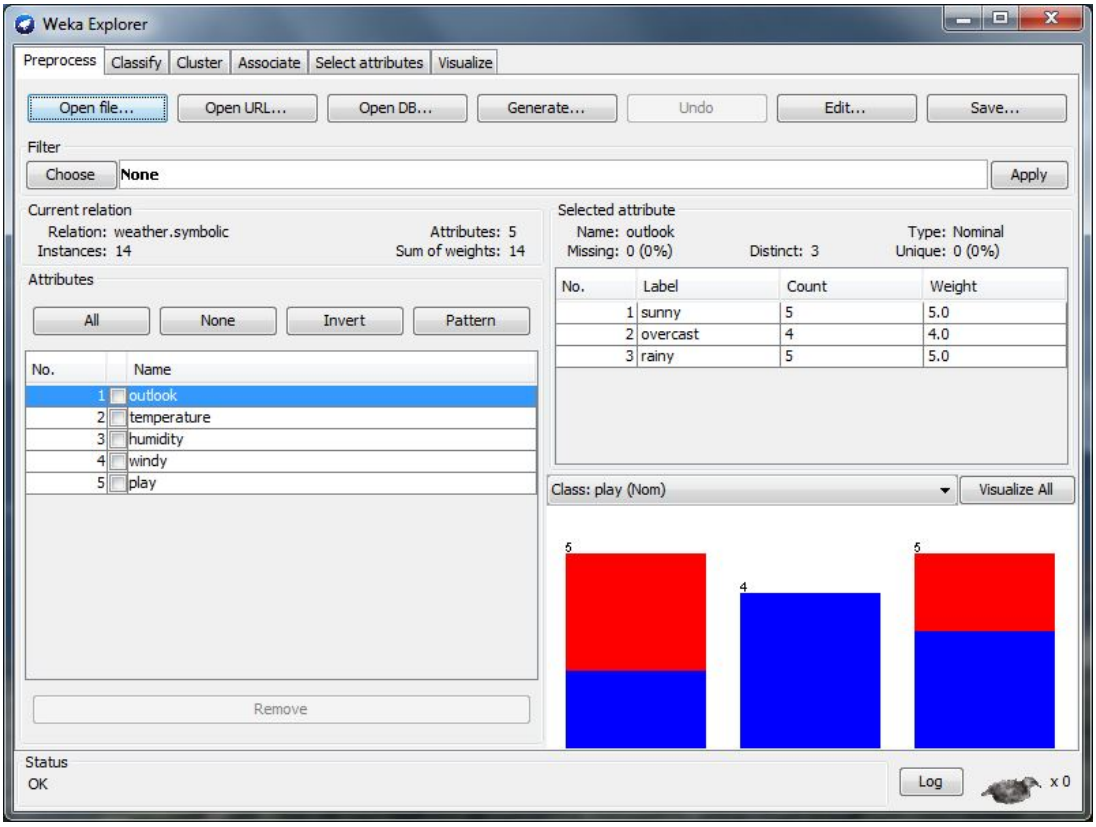
Panel *Cluster* slouží pro úlohy učení bez učitele, kde je hlavním cílem najít shluky. Tento panel obsahuje stejné možnosti jako panel *Classify*.

Panel *Associate* slouží k vytváření asociačních pravidel nad načteným datasetem. Obsahuje pouze možnosti výběru algoritmu a reprezentace výsledků.

Panel *Select attributes* poskytuje několik metod pro tuto problematiku. Ty zahrnují algoritmy pro hodnocení atributů a metodu hledání parametru.

Panel *Visualization* pomáhá vizualizovat načtená data v podobě grafické matice všech možných párů atributů. Zde Weka neprovádí žádné úlohy, nýbrž vizuálně pomáhá ve volbě vhodného přístupu k řešení dané úlohy.

Explorer je základní kámen nástroje Weka a slouží pro většinu analýz a úloh učení nad datasety. Nevýhodou tohoto rozhraní je načítání dat do OP počítače. Proto je Explorer vhodný pouze pro malé a středně velké datasety.



Obrázek 5: Explorer - panel preprocess nástroje Weka

3.5.2 Experimenter

Rozhraní *Experimenter* slouží pro pokročilou analýzu nad vstupními daty. Umožňuje například nad různými datasety testovat několik učících schémat s různými parametry a tak hledat řešení úlohy efektivněji než v Exploreru. Rozložení prvků rozhraní Experimenter lze vidět na obrázku 6.

Panel *Setup* nabízí rozhraní pro správu experimentu. Umožňuje vytvářet, ukládat a načítat nové experimenty spolu s nastavením, které datasety a jaké algoritmy a nastavení se má používat.

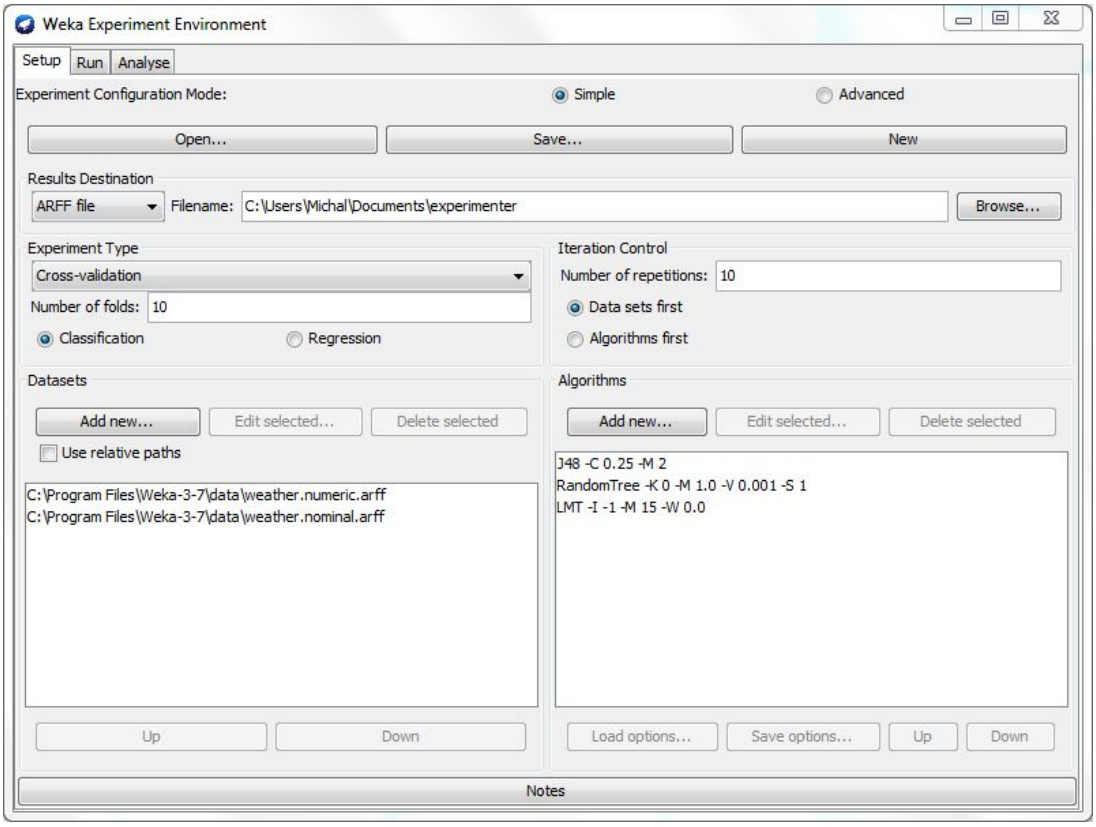
Panel *Run* obsahuje pouze možnost spuštění experimentu spolu s informacemi o průběhu.

Panel *Analyse* pak nad načteným experimentem dokáže provádět různé statistické testy a rovněž zobrazovat výsledky testování.

3.5.3 Knowledge Flow Interface

Jedná se o interpretaci *drag-and-drop* nástroje, ve kterém můžeme provádět stejné úkoly, jako v předchozích rozhraních. Výhodou tohoto rozhraní je fakt, že si můžeme definovat celý cyklus učení sami od počátečního načtení datasetu až po reprezentaci výsledků a ukládání. Takto můžeme ovlivňovat průběh, jakým tečou data skrze jednotlivé prvky.

Příklad použití KFI je uveden na obrázku 7. Cílem je klasifikační úloha s použitím algoritmu J48 a načtením dvou datasetů ve formátu CSV, přičemž jeden slouží jako tréninkový a druhý



Obrázek 6: Experimenter - panel Setup

jako testový. Výsledený model je pak ohodnocen z hlediska výkonosti a je zobrazen uživateli v textové formě.

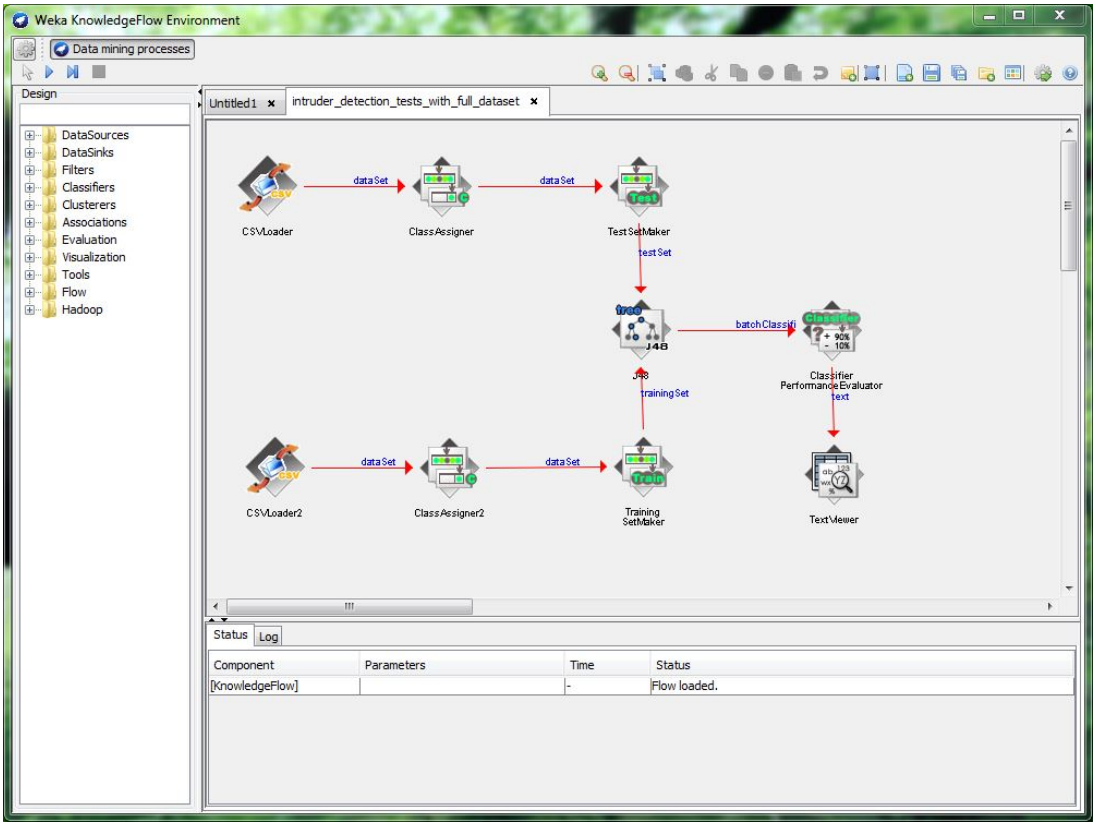
Obrovskou výhodou vytváření procesů v KFI je, že se při vytváření toku akcí nenačítá dataset do OP počítače, tudíž nám umožňuje používat v nástroji Weka velké datasety. Pokud ovšem vytvořený proces spustíme, bude Weka pracovat nad datasetem v OP počítače, což znova klade meze ve využití nad velkými daty. Tento problém můžeme vyřešit inkrementálním učením, které je v KFI dostupné.

3.5.4 Command Line Interface

Režim CLI nabízí pouze konzoli, kde lze provádět veškeré úlohy, jako v předchozích rozhraních. Výhoda tohoto přístupu je, že se můžou provádět veškeré úlohy tak, aniž by se muselo spouštět grafické rozhraní nástroje Weka.

Možnosti využití konzole lze posunout ještě o úroveň výš, pokud použijeme standardní konzoli operačního systému. Pomocí přepínačů můžeme spustit úlohu, aniž bychom vyvolali grafické rozhraní nástroje Weka. Zároveň tak ušetříme část OP počítače.

Jako příklad poslouží dávkový soubor, jehož obsah je znázorněn ve výpisu 2.



Obrázek 7: Proces v Knowledge Flow Interface

```
cd "C:\Program Files\Weka-3-7"
java -cp weka.jar weka.classifiers.meta.ClassificationViaRegression -W weka.classifiers.functions.LinearRegression -t
    data/iris.arff -x 2 -- -S 1
```

Výpis 2: Obsah dávkového souboru pro klasifikaci skrze systémovou konzoli

Spuštěním dávkového souboru provedeme klasifikaci podle zadaných parametrů. Část výsledku, zabývající se kvalitou klasifikace, můžeme sledovat na výpisu 3. Možnosti tohoto přístupu rozšiřují celkové využití nástroje Weka. Typická situace, ve které využijeme spuštění klasifikační úlohy skrze konzoli, je případ, kdy potřebujeme spustit úlohu na vzdáleném počítači, typicky s větším výkonem.

```
=== Stratified cross-validation ===

Correctly Classified Instances         123               82      %
Incorrectly Classified Instances        27               18      %
Kappa statistic                          0.73
Mean absolute error                      0.2349
Root mean squared error                  0.3157
Relative absolute error                 52.8443 %
Root relative squared error             66.9658 %
Coverage of cases (0.95 level)          98.6667 %
Mean rel. region size (0.95 level)      74.2222 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0,980    0,000      1,000   0,980      0,990    0,985    1,000     1,000     Iris-setosa
                 0,660    0,100      0,767   0,660      0,710    0,584    0,813     0,670     Iris-versicolor
                 0,820    0,170      0,707   0,820      0,759    0,629    0,938     0,865     Iris-virginica
Weighted Avg.    0,820    0,090      0,825   0,820      0,820    0,733    0,917     0,845

=== Confusion Matrix ===

  a  b  c   <-- classified as
 49  1  0 |  a = Iris-setosa
  0 33 17 |  b = Iris-versicolor
  0  9 41 |  c = Iris-virginica
```

Výpis 3: Část výsledku procesu klasifikace skrze režim CLI

3.5.5 Package Manager

Package Manager slouží v nástroji Weka pro instalaci a správu volitelných balíčků rozšíření nástroje Weka. Balíčky lze instalovat jak oficiální z centrálního repozitáře, tak neoficiální z vložené webové adresy, případně souboru. Balíčky mohou obsahovat prakticky jakékoliv rozšíření - od implementace nových algoritmů, přes implementace načítání jiného formátu vstupu, až po napojení na jiné technologie. Pokud se jedná o implementaci nových algoritmů, nové algoritmy se projeví ve všech režimech nástroje Weka. Pokud balíček obsahuje jiný typ rozšíření, slouží především pro použití v režimu KFI.

Příkladem takového rozšíření je balíček *distributedWekaHadoop*, zmíněný v kapitole 3.1. Tento balíček implementuje do režimu KFI nástroje pro práci s Hadoop technologií.

4 Detekce síťových útoků v nástroji Weka

V této kapitole se věnujeme reálnému využití nástroje Weka v problematice zvané *detekce síťových útoků* (často pojmenované jako *detekce průniků*, angl. *intrusion detection*). Naším cílem bylo provést analýzu datasetu z oblasti detekce průniků se zaměřením na demonstraci možností nástroje Weka.

4.1 Popis systému pro detekci síťových útoků

Průnik je definován jako soubor akcí, které se snaží narušit integritu, utajenost nebo dostupnost prostředků. *Detekce průniků* je proces monitorování a analýzy událostí, které se vyskytují v počítačových systémech (nebo sítích) [25]. Systém pro detekci síťových útoků (angl. Intrusion detection system) je program (nebo soubor programů), který analyzuje co se děje (nebo co se stalo) v klasickém provozu a hledá známky průniků. Tyto systémy slouží jako poslední obranný mechanismus pro zabezpečení systému.

Naštěstí je problematika detekce průniků známá a se stoupajícím výkonem počítačů a nalézáním nových metod je i řešení značně efektivnější než bylo na počátku.

4.1.1 Vstupní data

V roce 1999 byly pro potřeby soutěže uvolněny datasety, které zachycují síťový provoz obsahující útoky. Uvolněny byly jak datasety tréninkové, tak dataset testovací. Tréninkové datasety obsahují informaci o tom, zda-li se jedná o útok, či nikoliv (a pokud to byl útok, tak o jaký typ útoku se jedná). Jedná se o klasifikační úlohu učení s učitelem.

Pro řešení této problematiky jsme se rozhodli zvolit dataset *kddcup.data_10_percent* [26], který obsahuje pouze 10% záznamů původního datasetu (poměr tříd byl zachován). Učinili jsme tak z toho důvodu, že původní dataset obsahuje okolo 5 mil. záznamů a různé algoritmy strojového učení by zabraly neúměrnou dobu učením modelu.

Dataset obsahuje celkem 42 parametrů, kde prvních 41 parametrů jsou parametry síťového provozu a poslední parametr je třídní, určující typ útoku. Před zpracováním byla data normalizována (i kategoriální parametry). Původně dataset rozlišoval 24 typu útoků, ty byly ale pro potřeby experimentu shlukovány do pěti skupin (toto shlukování bylo již provedeno v rámci soutěže). Výslednou konfiguraci datasetu pro zpracování v nástroji Weka můžeme nalézt v tabulce 1.

Jelikož si některé vybrané algoritmy poradí pouze s binární klasifikací (tedy klasifikací do dvou tříd), byla data rozdělena celkem do pěti skupin. Každá skupina obsahuje data jednoho typu útoku a data, kde se o útok nejedná. Skupina *normal* obsahuje data normálního provozu. Takto bylo vytvořeno 5 skupin, každá obsahující tréninková a testová data. Rozdělení záznamů do skupin dle typu útoku můžeme vidět v tabulce 2. V nově vzniklých datasetech byl zachován poměr útoků vzhledem k celkovému počtu instancí, kromě datasetů *u2r_training* a *u2r_test*,

Tabulka 1: Parametry upraveného datasetu ze soutěže kdd cup

Původní dataset	
Dataset:	kddcup.data_10_percent
Počet záznamů:	494 021
Diskrétní parametry:	1, 2, 3, 6, 11, 20, 21
Spojité parametry:	0, 4, 5, 7-10, 12-19, 22-40
Třídní parametr:	41

Tabulka 2: Rozdělení záznamů pro jednotlivé typy útoků ve vytvořených datasetech problematiky detekce průniků

Tréninkové datasety					
Název datasetu:	normal_training	probe_training	dos_training	u2r_training	r2l_training
Počet záznamů:	5092	5092	5092	5092	5092
Z toho útoků:	1000 (19,63%)	500 (9,82%)	3002 (58,95%)	27 (0,53%)	563 (11,05%)
Testovací datasety					
Název datasetu:	normal_test	probe_test	dos_test	u2r_test	r2l_test
Počet záznamů:	6890	6890	6890	6890	6890
Z toho útoků:	1400 (20,32%)	700 (10,16%)	4202 (60,98%)	25 (0,36%)	563 (8,17%)

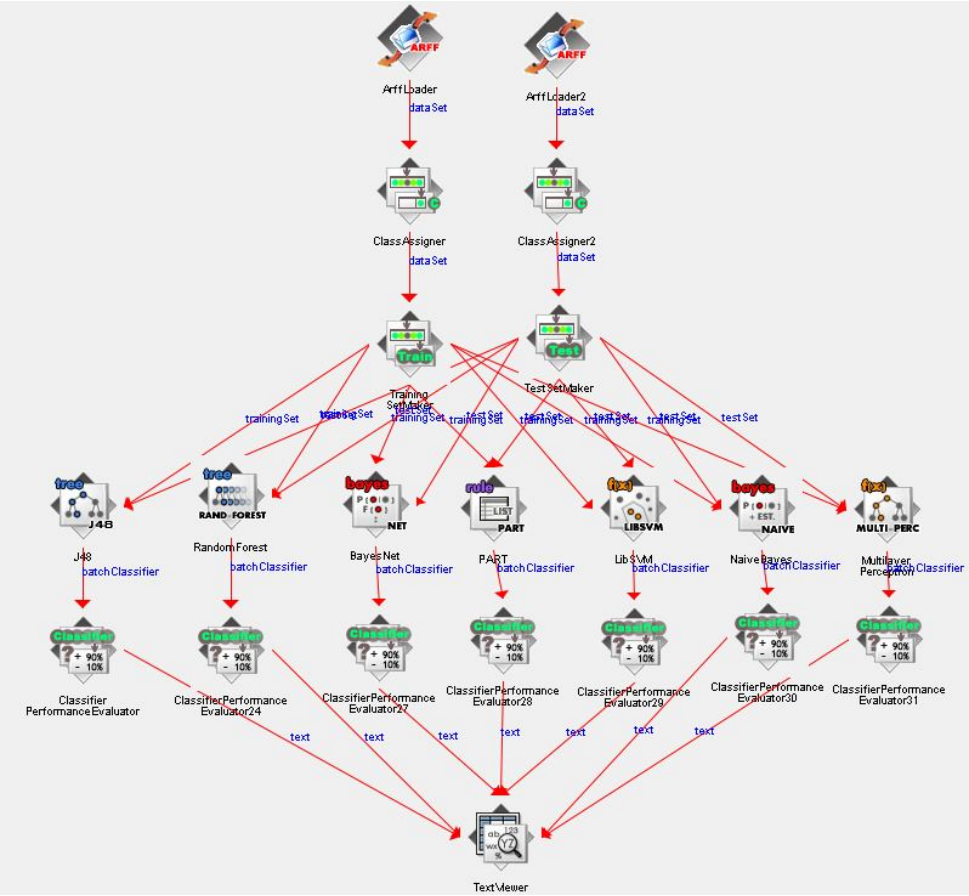
kde byly vyselektovány všechny útoky z dané skupiny (z důvodu velmi malého procentuálního zastoupení útoků).

4.1.2 Nastavení experimentu

Experiment jsme prováděli na osobním počítači s následující konfigurací: *Intel Core i5-3210M @ 2,5 GHz, RAM 8 GB DDR3 1333MHz, 1TB HDD 5400 ot/min., Windows 7 64bit.* Pokud není uvedeno jinak, všechny experimenty v této práci byly prováděny na tomto počítači.

Tento experiment byl inspirován pracemi Abrahama [28] a Valečka [29]. Použili jsme stejný vstupní dataset a rovněž jsme se inspirovali výběrem algoritmů.

Pracovali jsme v režimu KFI a vytvořili jsme takový proces, který nad vstupním datasetem provedl učení více algoritmů současně a rovněž vyhodnotil úspěšnost klasifikace jednotlivých modelů. Výsledný proces můžeme vidět na obrázku 8. Takto vytvořený proces paralelně načetl tréninková i testovací data, označil třídní atribut, provedl nad daty učení pomocí specifikovaných algoritmů, vyhodnotil jednotlivé modely a uložil výsledky. Tento proces lze jednoduše rozšířit o další algoritmy s libovolným nastavením. Režim KFI tudíž nabízí efektivní způsob, jak nad vstupním datasetem provést učení pomocí různých algoritmů. Pro účel experimentu jsme vybrali následující algoritmy:



Obrázek 8: Vytvořený proces v režimu KFI pro vícenásobnou klasifikaci vytvořených datasetů z problematiky detekce průniků

- Rozhodovací stromy - J48
- Asociační pravidla - PART
- Neuronové sítě - MP (multilayer perceptron)
- Shlukování - Simple k-means
- SVM

Veškeré algoritmy byly ponechány v původním nastavení, jelikož cílem experimentu nebylo najít nejoptimálnější řešení této problematiky, nýbrž demonstrovat možnosti nástroje Weka.

4.2 Výsledky experimentu

Měřítkem efektivity jsme považovali procentuální úspěšnost klasifikace a procento neodhalených útoků, jelikož je v této problematice nebezpečnější klasifikovat útok jako normální provoz, než v opačném případě vyvolat falešný alarm. V celkovém vyhodnocení nejvhodnějšího algoritmu

Tabulka 3: Výsledky klasifikace pro třídu *normal*

	k-means	**MP**	**SVM**	**J48**	**PART**
Počet záznamů:	6890				
Z toho útoků:	1400				
Úspěšnost klasifikace:	69,16%	97,8%	89,81%	**99,46%**	97,23%
Neodhalených útoků:	0 0%	**4** **0,28%**	512 36,57%	6 0,42%	**4** **0,28%**

Tabulka 4: Výsledky klasifikace pro třídu útoků *probe*

	k-means	**MP**	**SVM**	**J48**	**PART**
Počet záznamů:	6890				
Z toho útoků:	700				
Úspěšnost klasifikace:	52,67%	**99,85%**	95,66%	98,14%	98,08%
Neodhalených útoků:	298 42,57%	**4** **0,57%**	304 43,43%	5 0,71%	6 0,86%

jsme brali v úvahu následující kritérium: Pokud algoritmus odhalil většinu útoků, ale procentuální úspěšnost klasifikace je nízká, pak je metoda nevhodná pro danou problematiku.

Klasifikace třídy *normal* (tabulka 3) dopadla z hlediska procentuální úspěšnosti klasifikace v původním nastavení algoritmů nejlépe pro algoritmus J48 s úspěšností 99,46%. Algoritmus k-means sice odhalil všechny útoky, nicméně původní nastavení algoritmu vykazovalo nízkou celkovou úspěšnost klasifikace (69,16%.) Nejefektivnější detekci útoků provedly v původním nastavení algoritmy MP a PART s 0,28% neodhalených útoků. Algoritmus J48 neodhalil 0,42% útoků. Velice malou úspěšnost detekce průniků vykazoval algoritmus SVM s 36,57% neodhalených útoků.

Tabulka 4 ukazuje výsledky klasifikace pro třídu útoků *probe*. V procentuální úspěšnosti klasifikace algoritmů v původním nastavení si vedl nejlépe algoritmus MP s 99,85% přesností klasifikace. Algoritmus k-means vykazoval malou procentuální úspěšnost klasifikace (52,67%). Algoritmus MP neodhalil celkem 0,57% útoků. Algoritmy J48 a PART na tom byly podobně s 0,71%, resp. 0,86% neodhalených útoků. Algoritmus SVM rovněž vykazoval malou úspěšnost detekce průniků - 43,43% útoků klasifikoval jako normální provoz.

Tabulka 5 ukazuje výsledky klasifikace pro třídu útoků *dos*. Největší procentuální úspěš-

Tabulka 5: Výsledky klasifikace pro třídu útoků *dos*

	k-means	**MP**	**SVM**	**J48**	**PART**
Počet záznamů:	6890				
Z toho útoků:	4202				
Úspěšnost klasifikace:	75,19%	97,08%	91,51%	**99,56%**	98,27%
Neodhalených útoků:	1273 30,29%	199 4,73%	37 0,88%	**18** **0,43%**	96 2,28%

Tabulka 6: Výsledky klasifikace pro třídu útoků *u2r*

	k-means	**MP**	**SVM**	**J48**	**PART**
Počet záznamů:	6890				
Z toho útoků:	25				
Úspěšnost klasifikace:	50,82%	99,83%	99,63%	**99,91%**	99,77%
Neodhalených útoků:	24 96%	10 40%	25 100%	**3** **12%**	14 56%

Tabulka 7: Výsledky klasifikace pro třídu útoků *r2l*

	k-means	**MP**	**SVM**	**J48**	**PART**
Počet záznamů:	6890				
Z toho útoků:	563				
Úspěšnost klasifikace:	56,05%	**99,36%**	98,91%	99,33%	99,27%
Neodhalených útoků:	33 5,86%	**39** **6,92%**	44 7,82%	139 24,69%	50 8,88%

nost klasifikace algoritmů v původním nastavení vykazuje algoritmus J48 s hodnotou 99,56%. Algoritmus k-means měl procentuální úspěšnost klasifikace 75,19% a neodhalil celkem 30,29% útoků. Algoritmus J48 neodhalil pouze 0,43% útoků. Podobně si vedl algoritmus SVM s 0,88% neodhalených útoků, ovšem procentuální úspěšnost klasifikace byla velmi nízká.

V tabulce 6 můžeme vidět výsledky klasifikace pro třídu útoků *u2r*. Z hlediska procentuální úspěšnosti klasifikace algoritmů v původním nastavení dopadl nejlépe algoritmus J48 s úspěšností 99,91%. Ovšem i algoritmy MP, SVM a PART dosáhly úspěšnosti klasifikace přes 99,5%. Algoritmus k-means dosahoval procentuální úspěšnosti klasifikace pouze 50,82%. Z hlediska počtu neodhalených útoků vykazoval algoritmus J48 12% špatně klasifikovaných útoků. Ostatní algoritmy vyznačovaly značnou mírou chybovosti, kde algoritmus k-means špatně klasifikoval 100% útoků.

V tabulce 7 vidíme výsledky klasifikace pro třídu útoků *r2l*. V procentuální úspěšnosti klasifikace algoritmů v původním nastavení si nejlépe vedl algoritmus MP s úspěšností 99,36%. Algoritmy J48 a PART rovněž vykazovaly dobré výsledky s 99,33%, respektive 99,27% úspěšností. Algoritmus k-means dosahoval úspěšnosti klasifikace 56,05% a neodhalil 5,86% útoků, avšak kvůli nízkému procentu úspěšnosti klasifikace není tento výsledek kvalitní. Malé procento neodhalených útoků vykazoval algoritmus MP - 6,92%. Velkým procentem neodhalených útoků disponoval algoritmus J48 s 24,69% neodhalených útoků.

4.3 Shrnutí

Je patrné, že v původním nastavení zvolených algoritmů neexistuje algoritmus, který by jednoznačně předčil ostatní. Tato vlastnost je daná povahou vstupního datasetu. U útoků typu *normal*, *probe*, *dos* a *r2l* bylo v datasetu dostatek pozitivních instancí útoků pro úspěšnou klasifikaci. Nad takovými daty se vyznačoval největší přesností algoritmus ze skupiny NN. Ovšem i algorit-

mus ze skupiny DT vykazoval ve všech experimentech vysokou přesnost a díky své přímočarosti může být tou správnou volbou. Naopak v případě útoku typu *u2r* jsme se potýkali s velkou nerovnováhou vstupních dat. Konkrétně v případě *u2r* obsahoval tréninkový dataset pouze 27 (0,53%) instancí útoků. V takto nevybalancovaném datasetu s nedostatkem pozitivních instancí selhala většina algoritmů. Největší úspěšnosti dosáhl algoritmus ze skupiny DT - v průměru 72% úspěšnosti klasifikace. V tomto případě by bylo vhodné vstupní dataset předzpracovat a zvýšit počet pozitivních instancí tak, aby výsledná klasifikace dosahovala výsledků jako u ostatních typu útoků. Nástroj Weka nabízí možnosti předzpracování vstupního datasetu v režimu Explorer, popsaného v kapitole 3.5.1.

V experimentu jsme se zaměřili na práci v nástroji Weka nad datasetem z oblasti detekce průniků. Pracovali jsme v režimu KFI, ve kterém jsme byli schopni vytvořit proces, který provedl klasifikaci pomocí různých algoritmů strojového učení. Následně jsme popsali výsledky jednotlivých klasifikací. Na závěr jsme uvedli způsob, díky kterému bychom pomocí nástroje Weka dosáhli kvalitnější klasifikace.

Nástroj Weka je vhodný pro hledání řešení podobných problémů, kde dataset bezproblémově načteme do OP počítače, tudíž můžeme provádět analýzu nad takovým datasetem. Uživatelské rozhraní nástroje Weka je příjemné a intuitivní. Nástroj Weka rovněž poskytuje velké množství algoritmů pro řešení různých problémů.

5 Struktura nástroje Weka

V této kapitole se věnujeme nástroji Weka z technického hlediska. Zajímá nás, jak je nástroj Weka strukturován - jak spolu komunikují jednotlivé moduly, jaké druhy algoritmů Weka nabízí, jaké jsou jejich rozdíly a jaké je jejich potencionální využití. Dále analyzujeme, jak nástroj Weka správně rozšířit o vlastní algoritmy a metody. Také se zabýváme tím, jak práci v nástroji Weka paralelizovat. V neposlední řadě jsou porovnány výhody a nevýhody, které plynou z použité platformy Java™. Na závěr je ukázáno, jak rozšířit nástroj Weka o volitelné balíčky.

5.1 Modularizace

Nástroj Weka se skládá z několika modulů, které spolu úzce souvisí a jsou na sobě závislé. Jednotlivé moduly odpovídají jednotlivým částem procesu při typické úloze klasifikace v nástroji Weka. Zjednodušeně můžeme vidět tento proces klasifikace na obrázku 9. Z celkového procesu klasifikace extrahujeme základní moduly:

- **Načítání dat** - modul se stará o načtení datasetu [1] do OP počítače z dostupného úložiště. Poskytuje data následujícímu modulu.
- **Filtrace dat** - modul přijímá dataset nebo instanci, kde provádí různé předzpracování pro následné učení. Výstupem je předzpracovaný dataset nebo instance.
- **Modul učení** - modul přijímá dataset nebo instanci, na kterou aplikuje zvolený algoritmus. Výstupem tohoto modulu je naučený model klasifikace.
- **Hodnocení modelu učení** - modul přijímá naučený model klasifikace, na který aplikuje evaluační metriku. Výstupem je hodnocení modelu učení.
- **Ukládání dat** - modul se stará o uložení výsledné klasifikace.
- **Vizualizace dat** - na základě hodnocení klasifikace modul vizualizuje výsledky do různých podob.

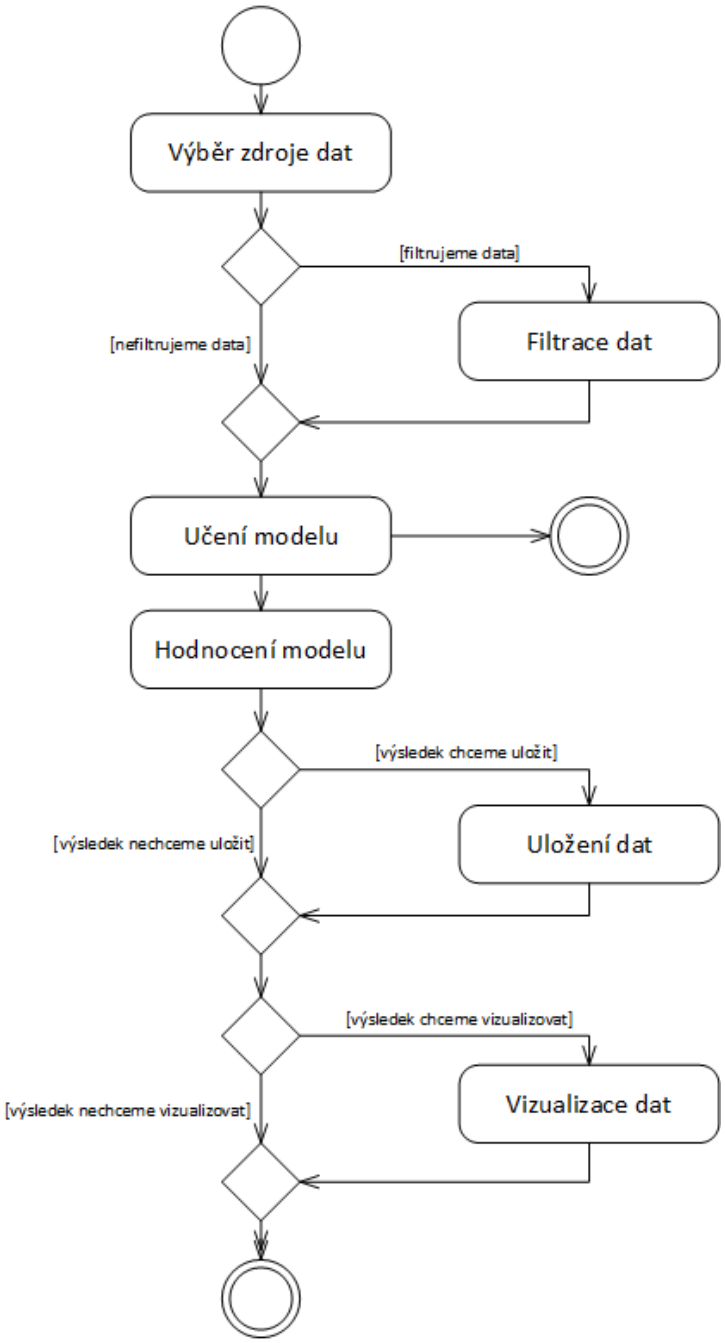
Toto rozdělení nemusí být na první pohled zřejmé při klasifikaci v režimu Explorer, nicméně v režimu KFI lze toto členění vidět na ukázkovém procesu znázorněném na obrázku 10.

V ukázkovém procesu režimu KFI figuruje oproti standardnímu procesu navíc položka *TrainTestSplitMaker*, která je pomocná a rozděluje vstupní dataset na tréninkovou a testovací část. Zároveň v tomto procesu můžeme vidět zmíněný tok dat mezi jednotlivými moduly.

Nástroj Weka je tedy rozdělen do několika modulů. Je důležité uvědomit si tuto strukturu při úvaze, zda je tento nástroj vhodný pro naše použití.

Jelikož jsou zdrojové kódy nástroje Weka volně dostupné ke stažení, můžeme se dále ponořit do specifičtějších částí procesu.

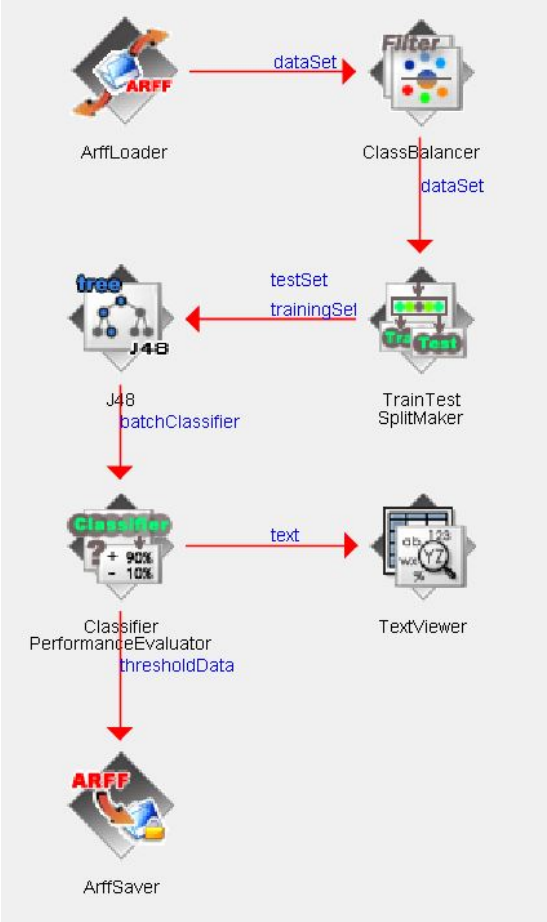
[1] jednotlivých instancí u inkrementálního učení



Obrázek 9: Standardní proces klasifikace v nástroji Weka

5.2 Algoritmy učení

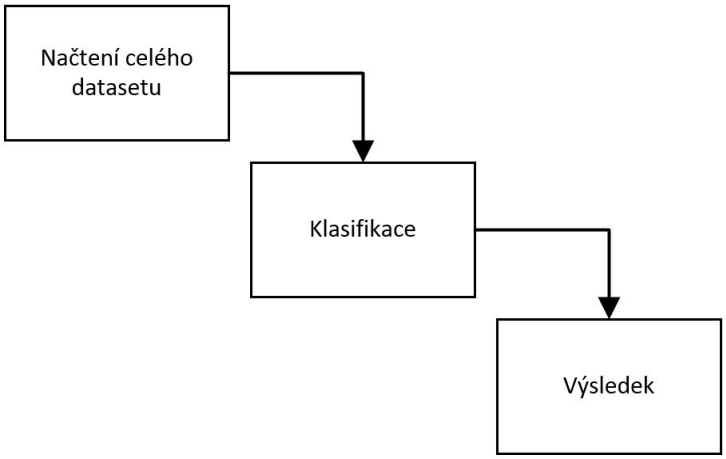
Základním prvkem, který ovlivňuje výslednou kvalitu klasifikace, je kvalita zvoleného algoritmu, který stojí za procesem učení modelu. V nástroji Weka se setkáváme se dvěma druhy algoritmů:



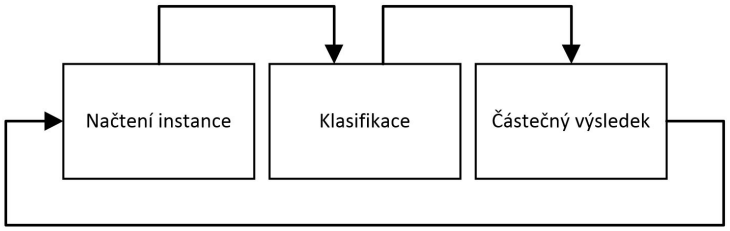
Obrázek 10: Komunikace modulů v režimu KFI

- **Standardní** - algoritmy, které pracují s celým datasetem. Vyžadují načtení celého datasetu do OP počítače. Učení modelu probíhá najednou.
- **Inkrementální** - klasifikátory, které pracují nad jednotlivými instancemi. Nevyžadují načtení datasetu do OP počítače. Učení modelu probíhá iterativně.

Na obrázku 11 vidíme proces učení modelu pomocí standardního algoritmu. Výhodou tohoto zpracování je přímočarejší implementace oproti inkrementálním algoritmům. Takový algoritmus pak může pracovat ve více vláknech a zpracovávat celý dataset efektivněji. Rovněž je dostupné větší množství algoritmů pro standardní zpracování. Tento model je velice podobný vodopádovému modelu používaném při vývoji software. Má také podobné nevýhody - ta největší je, že výsledek obdržíme až po celkové klasifikaci. Na obrázku 12 vidíme proces učení inkrementálního algoritmu. Toto zpracování má výhodu v tom, že v průběhu učení dostáváme částečný výsledek. Největší výhodou je ovšem absence nutnosti načítat dataset do OP počítače. Dataset se načítá inkrementálně po instanci a takto se algoritmus postupně zdokonaluje. Tento model se podobá agilnímu modelu používánému při vývoji software, kde každou další iterací dostáváme další část software.



Obrázek 11: Průběh učení pomocí standardního algoritmu



Obrázek 12: Průběh učení pomocí inkrementálního algoritmu

5.2.1 Rozdíly

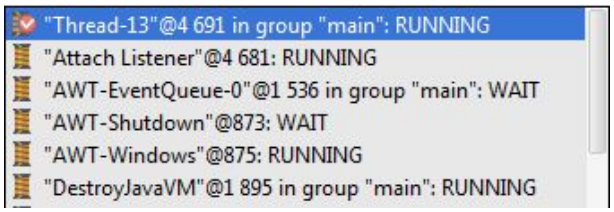
Rozdíly mezi těmito druhy algoritmů se projevují převážně v efektivitě při různé velikosti vstupního datasetu. Zatímco se standardní algoritmy převážně vyplatí používat na menší a středně velké datasety, inkrementální algoritmy nachází své využití zejména u velkých datasetů. Najít hranici velikosti datasetu, na kterém je již vhodnější učit se pomocí inkrementálního algoritmu, není triviální úkol, jelikož záleží na mnoha faktorech. Hlouběji je tato problematika rozebrána v kapitole 6.

Další rozdíl, který může výrazně ovlivnit kvalitu klasifikace (převážně z hlediska času učení), je implementace algoritmu. Jak jsme již zmínili, standardní algoritmy nabízí větší možnosti paralelizace procesu učení, než je tomu u inkrementálních algoritmů.

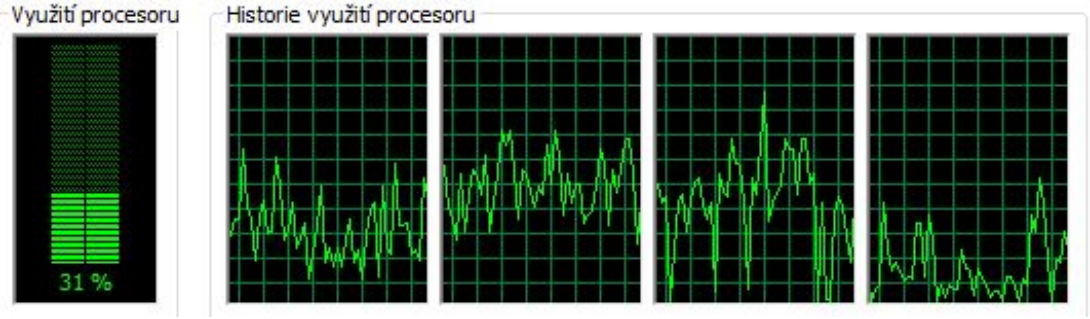
5.2.2 Implementace v nástroji Weka

V nástroji Weka se setkáváme se širokou základnou algoritmů strojového učení. Tato kapitola je zaměřena na implementaci algoritmů, které jsme již použili v problematice detekce průniků v kapitole 4.

Po detailní analýze zdrojových kódů jednotlivých algoritmů jsme zjistili, že algoritmy použité v kapitole 4 disponují pouze jednovláknovou implementací, jak můžeme vidět na obrázku 13. Rovněž tak algoritmy nevyužívají naplno možnosti procesorů (viz obrázek 14). Tohle tvrzení se



Obrázek 13: Využití vláken algoritmu J48 v nástroji Weka



Obrázek 14: Využití procesorů algoritmu J48 v nástroji Weka

ovšem nevztahuje na všechny algoritmy, jejichž implementaci nástroj Weka nabízí. Například algoritmus *Random Forest* disponuje mnohovláknovou implementací, kde si dokonce můžeme z uživatelského pohledu zvolit počet vláken, jež má algoritmus využívat. Takto implementované algoritmy pak převážně efektivněji využívají systémové prostředky.

Pro zefektivnění procesu klasifikace jsme analyzovali dvě řešení. První řešení je rozšíření nástroje Weka o další algoritmy, jež mohou danou problematiku řešit lépe. O možnosti rozšíření nástroje Weka z hlediska přidávání a úpravy algoritmů píšeme v kapitole 5.3. Druhé řešení se zaměřuje na paralelizaci procesu klasifikace v nástroji Weka. Tento přístup je popsán v kapitole 5.5.

5.3 Vlastní rozšíření nástroje Weka

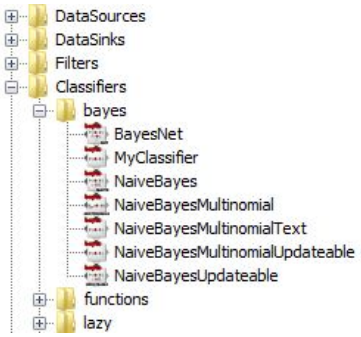
Nástroj Weka je napsaný v jazyce Java a díky opensource distribuci si může každý nástroj Weka optimalizovat ke svému užití. Nástroj Weka nabízí široké možnosti rozšíření převážně v implementaci vlastních algoritmů, musíme ovšem dbát určitých konvencí. Rozšiřovat můžeme všechny moduly uvedené v kapitole 5.1. Níže uvedený příklad naznačuje rozšíření nástroje Weka o vlastní algoritmus.

Z důvodu zachování funkcionality je důležité, abychom zdrojový kód vlastního algoritmu vložili do správného balíčku. Pokud bude náš algoritmus patřit do skupiny bayesovských algoritmů, musí být zdrojové kódu umístěny v balíčku *weka.classifiers.bayes* z důvodu použité reflexe. Zároveň musí mít třída strukturu znázorněnou ve výpisu 4.

```
package weka. classifiers .bayes;

public class MyClassifier extends AbstractClassifier implements OptionHandler{

    @Override
    public void buildClassifier (Instances data) throws Exception {
    ...
    }

    ...
}
```

Výpis 4: Definice třídy vlastního algoritmu

Kde povinná je implementace základního rozhraní *weka.classifiers.Classifier*. Nástroj Weka nabízí několik abstrakních tříd dle typu algoritmu, které toto rozhraní implementují. Námi vytvořený algoritmus rozšiřuje *AbstractClassifier*, který nabízí nejjednodušší implementaci. Volitelné jsou implementace dalších rozhraní, která přidávají našemu algoritmu další funkcionalitu. Po úspěšné implementaci zkompilujeme aplikaci a po spuštění aplikace již můžeme vidět náš algoritmus (viz obrázek 15) v nabídce klasifikátorů. Takto vytvořený algoritmus je dostupný ve všech režimech práce (Explorer, Experimenter, KFI a CLI). Podrobný návod, jak přidat vlastní algo-



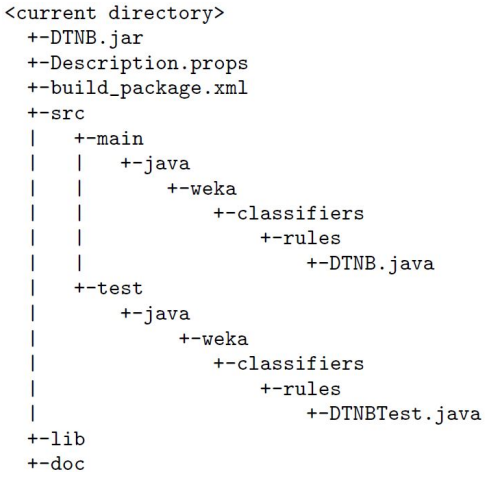
Obrázek 15: Vlastní klasifikátor v nabídce algoritmů režimu KFI

ritmus (nebo jiný modul, například zdroj dat) do nástroje Weka je detailně znázorněn v manuálu [30].

Ve vlastní implementaci algoritmu nám již nic nebrání použít veškeré dostupné možnosti platformy Java pro efektivní implementaci (včetně využití mnohovláknového zpracování). Také můžeme tímto způsobem implementovat již napsané algoritmy v jiných jazycích s použitím JNI.

5.3.1 Distribuce vlastního rozšíření

Pokud bychom chtěli náš vyvinutý algoritmus, zdroj dat nebo například celý vlastní modul dále šířit, nabízí se řešení skrze tzv. *Weka balíčky*.



Obrázek 16: Struktura balíčku DTNB

Weka balíček je zip archiv, který obsahuje třídy, kterými chceme nástroj Weka rozšířit a zároveň zachovává definovanou strukturu. Příklad balíčku s algoritmem DTNB můžeme vidět na obrázku 16.

Návod, co vše musí takový balíček krom samotného algoritmu obsahovat, můžeme nalézt v manuálu. Takto vytvořený balíček můžeme skrze *package manager* instalovat do nástroje Weka.

5.4 Rozšíření nástroje Weka pomocí dostupných balíčků

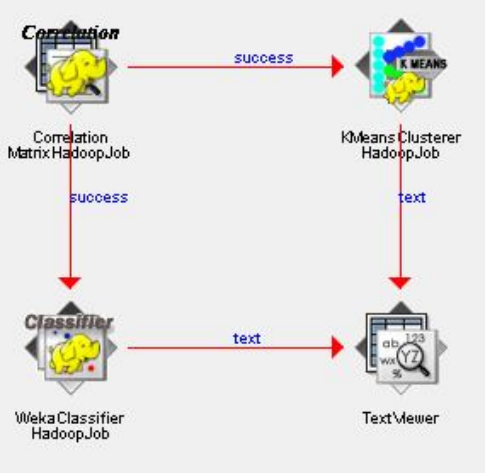
Pomocí *package manageru* (kapitola 3.5.5) můžeme spravovat veškeré balíčky v nástroji Weka. V následujících kapitolách jsou uvedeny jednotlivé druhy balíčků, se kterými se při práci v nástroji Weka můžeme setkat.

5.4.1 Balíčky obsahující implementaci nových algoritmů

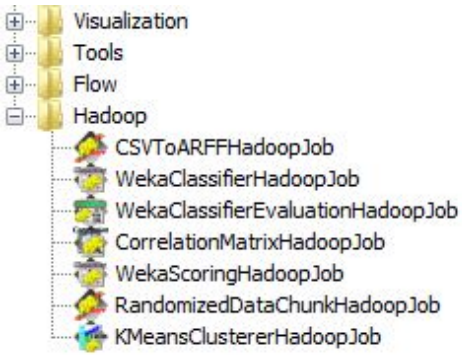
Tento druh balíčků implementuje do nástroje Weka jednotlivé algoritmy. Struktura takového balíčku je podobná jako na obrázku 16. Důležitou vlastností je, že takto přidaný algoritmus spadá do již existujícího modulu. Příklad takového balíčku je například *libSVM*[31], který implementuje algoritmus SVM do nástroje Weka.

5.4.2 Balíčky obsahující nový modul

Tento druh balíčků se od předchozího liší tím, že implementuje do nástroje Weka celý nový modul, který se může stát novou částí procesu, nebo určitou část nahradit. Příkladem je balíček *Weka-Parallel*, který nahrazuje stávající modul evaluace algoritmů a nahrazuje jej celkovou vlastní implementací. Projektu *Weka-Parallel* je věnována kapitola 5.5.2.1



Obrázek 17: Proces v KFI s použitím balíčku *distributedWekaHadoop*



Obrázek 18: Balíček *distributedWekaHadoop* v kontextové nabídce režimu KFI

5.4.3 Balíčky implementující celý proces

Narozdíl od předchozích balíčků, tento balíček přidává implementaci celého vlastního procesu klasifikace. Příkladem je balíček *distributedWekaHadoop* [20], který implementuje práci s technologií Hadoop. Na obrázku 17 vidíme proces v režimu KFI, který je vytvořen z modulů tohoto balíčku. Obrázek 18 znázorňuje nový balíček v kontextové nabídce režimu KFI. Tento balíček se stará o celkový proces načtení dat, klasifikace a evaluaci výsledného modelu. Výstup je poté realizován standardním výstupem.

5.5 Paralelizace

Z dosavadních znalostí nástroje Weka se nám jeví několik míst, které bychom mohli efektivně paralelizovat tak, abychom dosáhli větší efektivity výsledné klasifikace. Větší efektivitou zde myslíme optimalizaci z hlediska časové náročnosti učení modelu a následné evaluace. Následující sekce popisují jednotlivé možnosti.

5.5.1 Paralelizace algoritmů

Tento přístup k paralelizaci vyžaduje hlubší znalost platformy Java a také programování. Jedná se o přístup, ve kterém se snažíme vytvořit nejkvalitnější implementaci algoritmu. V dnešní době vícejádrových procesorů je nejefektivnější variantou vícevláknové zpracování.

Při uvážení paralelizace algoritmu je třeba brát v potaz povahu zvoleného algoritmu - zda-li je vůbec paralelní zpracování datasetu možné. Pokud ano, můžeme vhodnou implementací výrazně snížit dobu učení. Tento způsob paralelizace je jediný, který dokáže ovlivnit samotnou dobu učení. Jelikož je tento přístup přímočarý, rozhodli jsme se nezabývat ukázkovou implementací (která je již nastíněna ve výpisu 4).

5.5.2 Paralelizace procesu klasifikace

Paralelizace celého procesu klasifikace nabízí jiný způsob paralelizace. Zaměřuje se především na zefektivnění jednotlivých částí procesu klasifikace.

Příkladem může být úvaha, zda-li se dá nějakým způsobem paralelizovat zpracování datasetu. Mějme zdroj dat daný souborem (například textovým) s cílem klasifikovat jednotlivé instance. V nástroji Weka můžeme k učení přistoupit několika způsoby, znázorněnými na obrázku 19:

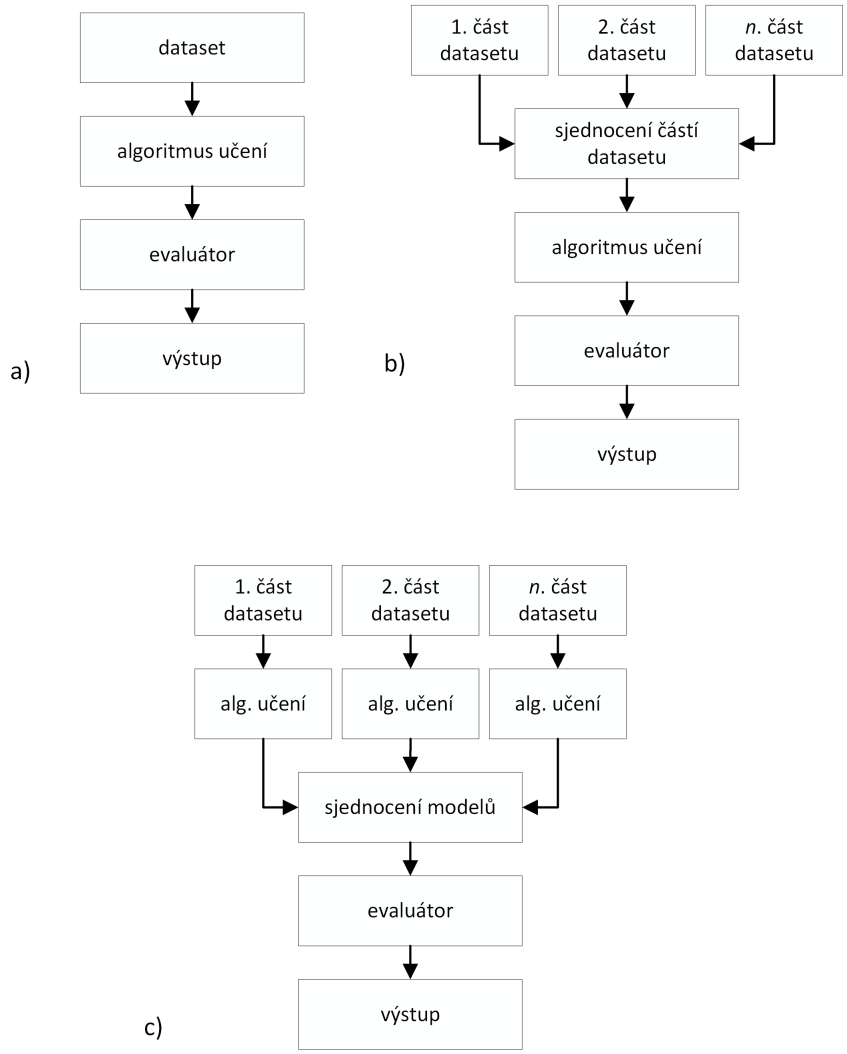
- **bez paralelizace** - varianta a) obrázku. Soubor standardně načteme do OP, klasifikujeme, evaluujeme model a vypíšeme výstup. V tomto případě proces klasifikace neparalelizujeme a nástroj Weka nijak neupravujeme.

- **paralelizace načítání** - varianta b) obrázku. Soubor je před načtením rozdělen na několik částí. Do nástroje Weka musíme implementovat logiku, která dokáže spojit jednotlivé části datasetu a předat jej algoritmu učení. Proces klasifikace pokračuje dále standardně. Tato varianta neparalelizuje samotný proces učení, pouze proces načítání vstupních dat.

- **paralelizace načítání a učení** - varianta c) obrázku. Soubor je před načtením rozdělen na několik částí, kde každou část zpracovává samostatný algoritmus. Výstupy jednotlivých algoritmů jsou pak sjednoceny a zpracování postupuje dále standardním postupem. V tomto případě musíme implementovat logiku, která dokáže sjednotit modely algoritmů. Tato varianta paralelizuje proces načítání a učení modelu.

Tento způsob paralelizace je také znám pod pojmem *embarassingly parallel*. V tomto případě se zaměřujeme na paralelizaci jednotlivých částí (modulů) procesu klasifikace.

5.5.2.1 Weka-Parallel V předchozí kapitole jsme uvedli paralelizaci dvou částí klasifikačního procesu - načítání vstupního datasetu a učení modelu. V procesu klasifikace existuje ještě jedno slabé místo z hlediska časové náročnosti celkového učení - evaluace modelu. Pro evaluaci se hojně používá metoda křížové validace, jakožto spravedlivá evaluační metrika naučeného modelu. Křížová validace je detailněji popsána v kapitole 3.4.4. Velká časová náročnost křížové validace



Obrázek 19: Návrhy paralelizace procesu klasifikace v nástroji Weka

vzniká při postupném testování jednotlivých skupin, přičemž způsob testování je pokaždé stejný. Tohoto problému si všiml Celis a Musicant [23], kteří vymysleli způsob, jak křížovou validaci distribuovat.

Weka-Parallel, jak své rozšíření Celis a Musicant nazvali, nabízí prostředky pro distribuci jednotlivých skupin křížové validace na další počítače. Uživatel, který chce tuto možnost využít, si musí stáhnout spustitelný archiv [32]. Tento archiv není standardním rozšířením nástroje Weka, ovšem jeho náhradou - jedná se o vlastní verzi, která disponuje implementací distribuované křížové validace, agregace výsledků jednotlivých částí a meziprocesovou komunikací. Uživatel se tak nemusí starat o to, jak se samotná distribuce skupin provádí.

Abychom mohli distribuované křížové validace pomocí *Weka-Parallel* dosáhnout, musíme splnit následující kritéria:

- možnost spuštění archivu *Weka-Parallel* jako démon OS na vzdáleném počítači (dále nazýváme server) - na serveru se inicializuje spojení typu Socket [33]
- jednotlivé servery musí být v dosahu sítě s počítačem, který provádí klasifikaci (dále nazýváme klient) - je nutné, aby se klient se servery spojil prostřednictvím spojení typu Socket

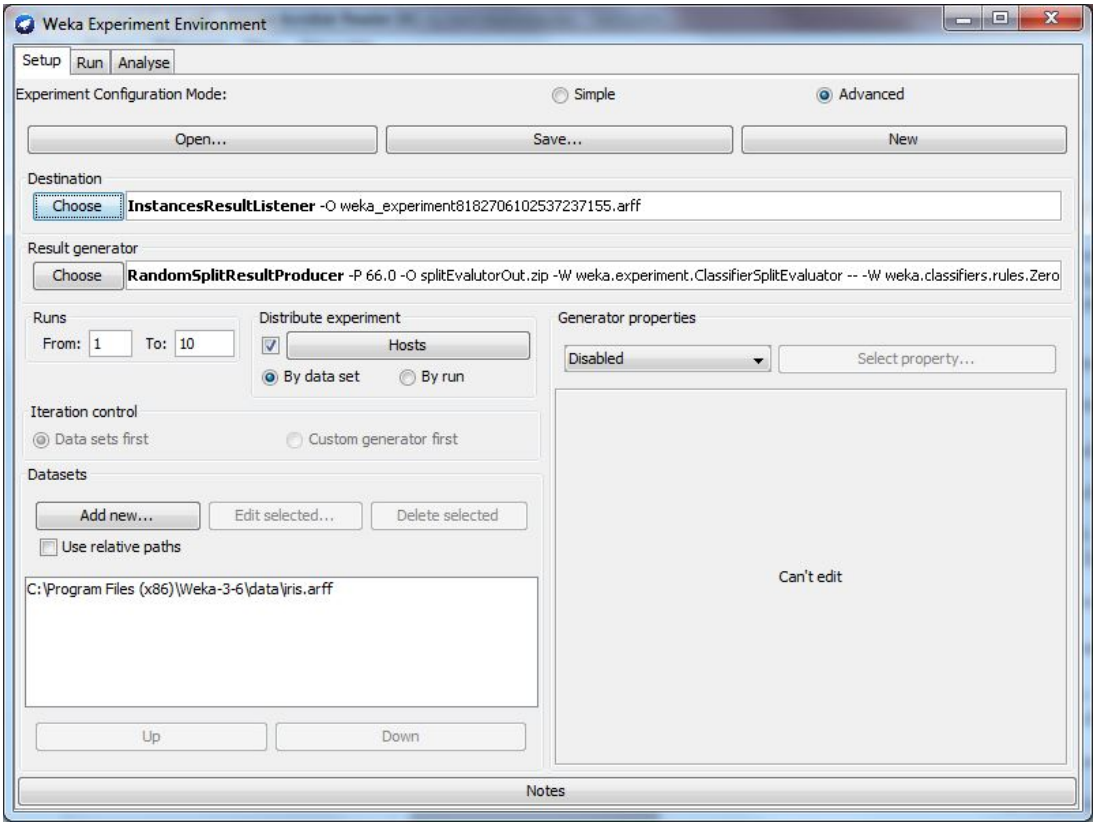
Jakmile splníme tyto podmínky, správným nastavením podle návodu [23] můžeme docílit posílání jednotlivých skupin na servery. Klient k tomu využívá *round-robin* algoritmus tak dlouho, dokud nejsou všechny skupiny vypočtené. Jednotlivé servery musí mít přístup ke vstupnímu datasetu. Pokud přístup nemají, dataset se posílá skrze spojení mezi klientem a serverem. V případě velkého datasetu může zde nastat problém, kde přenesení datasetu skrze TCP spojení může trvat déle, než samotná křížová validace na samostatném počítači (nehledě na prostorovou náročnost). Každá skupina se na serveru počítá v jednom vlákně. Pokud máme k dispozici server s více procesory a chceme využít naplno možností všech procesorů - můžeme na jednom serveru počítat více skupin najednou. Trik spočívá v tom, že v konfiguračním souboru na klientovi, kde se nastavuje spojení a kde se zadávají adresy jednotlivých serverů, duplikujeme adresu serveru, na kterém chceme počítat více skupin. Tímto způsobem dokážeme v extrému spustit křížovou validaci paralelně na jediném serveru, který může být zároveň i klientem (bohužel, podmínka na spojení typu Socket trvá).

5.5.2.2 Distribuované učení v režimu Experimenter Ačkoliv je přístup k paralelizaci křížové validace pomocí *Weka-Parallel* velice efektivní, vychází ze samotných možností nástroje Weka. Podobnou funkcionalitu můžeme hledat v režimu *Experimenter* v *advanced* módu, znázorněném na obrázku 20. Zde nám nástroj Weka nabízí možnost distribuovaného učení. Musíme ovšem splnit následující kritéria:

- máme k dispozici databázový server, ke kterému mají všechny vzdálené stroje (servery) přístup
- na klientovi máme v domovské složce spustitelný archiv nástroje Weka (weka.jar)
- na klientovi i serveru máme k dispozici spustitelné archivy konektorů na databázi
- na serverech máme právo spustit program jako démon

Detailní popis, jak experiment provést, je k dispozici v návodu [30].

Distribuované učení pomocí režimu Experimenter nese řadu omezení na použité počítače. Ačkoliv bychom přístupem k paralelizaci pomocí *Weka-Parallel* nebo režimu *Experimenter* dosáhli vysoké úrovně celkové paralelizace, je tento přístup v řadě případů obtížný implementovat (zejména tam, kde nemáme administrátorská práva nad systémem a jednoznačnou identifikaci v síti).



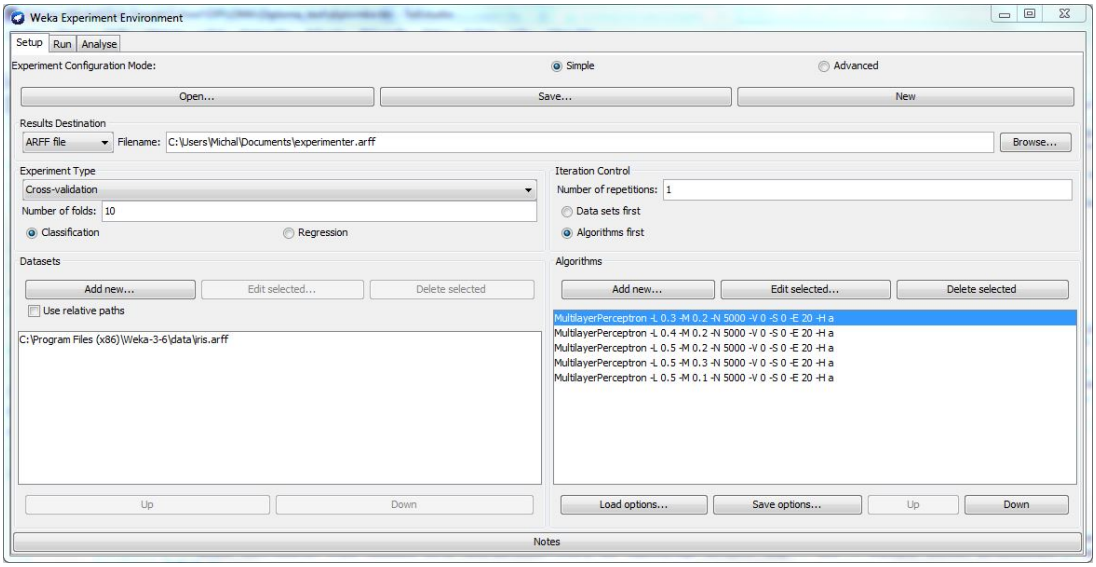
Obrázek 20: Experiment s nastavenou distribuovanou klasifikací v režimu Experimenter

5.5.3 Paralelizace nástroje

Mnoho problémů z oblasti strojového učení se zabývá nejen kvalitou učení vybraného algoritmu, ale také hledání optimálních parametrů zvoleného algoritmu (případně hledání vhodného algoritmu pro řešení dané problematiky). V takovém případě není efektivní paralelizace konkrétního algoritmu nebo konkrétní části procesu klasifikace tak přínosná, jako možnost paralelizovat celý nástroj a zvolit odlišné algoritmy a jiné parametry. Jelikož je nástroj Weka napsán v jazyce Java a disponuje možností spuštění nástroje (potažmo konkrétní úlohy) přímo z příkazové řádky, nabízí se zde možnost spustit proces hledání vhodného parametru/algoritmu paralelně. Toho je docíleno tím, že každá spuštěná aplikace má v OP počítače vyhrazený svůj vlastní JVM.

Nástroj Weka disponuje v režimu *Experimenter* stejnou možností hledání vhodného parametru. V panelu *Setup* můžeme definovat datasety a algoritmy (s nastavením parametrů), které chceme testovat. Tento experiment má oproti navrhované variantě nevýhodu - nefunguje paralelně. Každá možnost je spuštěna až jakmile skončí učení modelu pomocí předchozího algoritmu. Výhodou je ovšem řízený výstup do jediného souboru spolu se souhrnnými statistikami, nad kterými lze provádět statistické testy pomocí nástroje Weka.

Na příkladu jsou demonstrovány oba zmíněné přístupy k hledání vhodného nastavení parametrů učícího algoritmu. Mějmě daný vstupní dataset se 150 záznamy, pro který chceme najít



Obrázek 21: Hledání vhodného parametru v režimu Experimenter - vícenásobné nastavení algoritmu multilayer perceptron

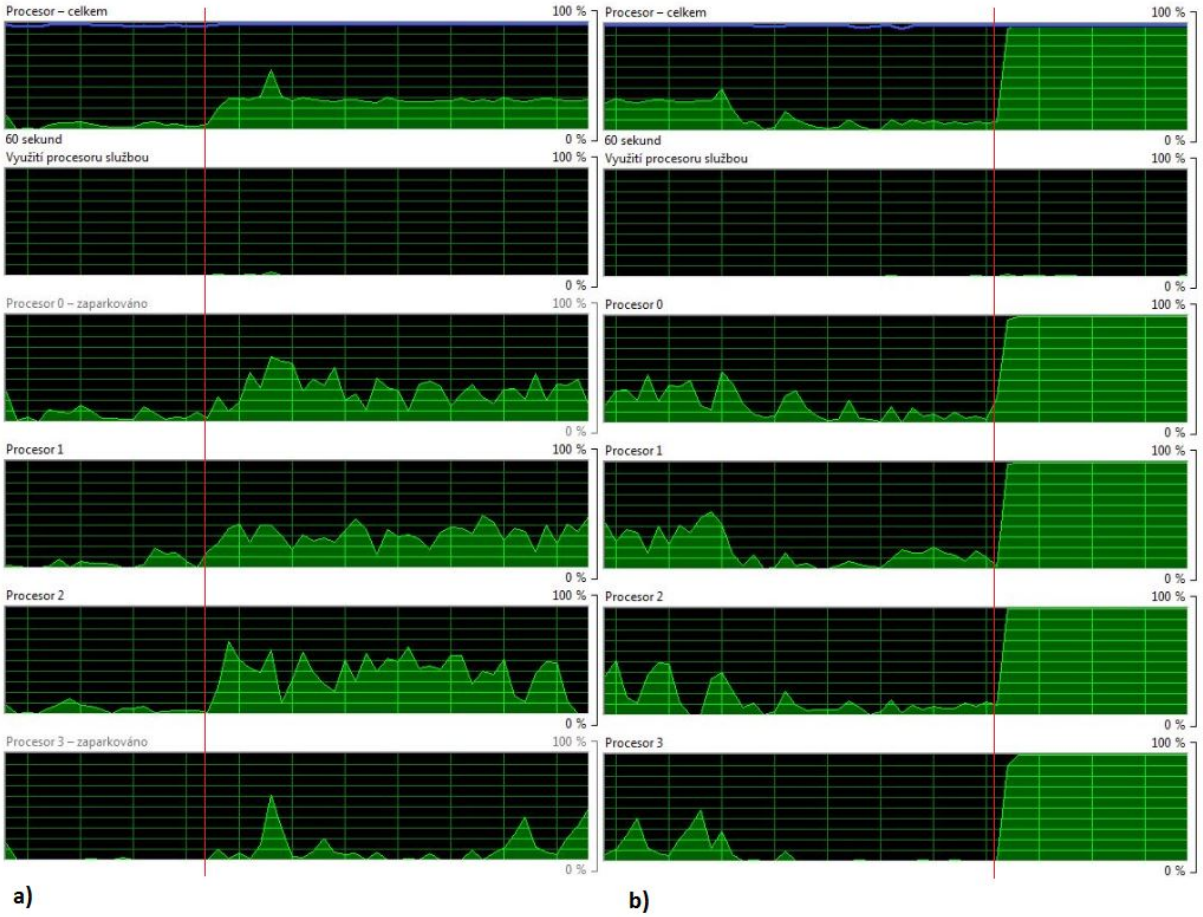
Tabulka 8: Porovnání procesů hledání vhodných parametrů algoritmu *multilayer perceptron* v režimu Experimenter a dávkového souboru

Režim	**Čas učení**	**Využitá operační paměť**
Experimenter	35 s	162 MB
dávkový soubor	22 s	115 MB

vhodné parametry algoritmu *multilayer perceptron*. Na obrázku 21 vidíme nastavené parametry algoritmu, ze kterých chceme najít nejvhodnější variantu.

V tomto nastavení jsme spustili experiment v režimu *Experimenter*. Zároveň jsme vytvořili dávkový soubor pro systém Windows, který spustil každou variantu skrze vlastní instanci nástroje Weka. Použili jsme k tomu příkaz `start javaw [...]`. Experiment jsme spustili na osobním počítači (konfigurace je popsána v kapitole 4.1.2). Výsledky porovnání v tabulce 8 poukazují na rychlejší proces klasifikace u dávkového souboru a to zhruba o třetinu z důvodu paralelizace. Rovněž využitá OP počítače byla menší u dávkového souboru. To je ovšem dáno tím, že v dávkovém režimu byly příkazy spuštěny bez grafického rozhraní. Na obrázku 22 můžeme vidět porovnání využití procesorů u obou variant. Varianta a) reprezentuje režim *Experimenter*, varianta b) dávkový soubor. Vertikální červená čára znázorňuje spuštění procesu klasifikace. Rozdíl je patrný - varianta dávkového souboru mnohem efektivněji pracovala s procesory. To je dáno právě paralelním zpracováním. Znatelnější rozdíl bude patrný při zpracování nad větším datasetem nebo nad časově náročnějším klasifikátorem.

Tento experiment potvrzuje, že nástroj Weka je pro problematiku hledání vhodného parametru (analogicky také algoritmu) vhodný. Rovněž také navrhnutý způsob paralelizace znatelně



Obrázek 22: Porovnání využití procesorů u režimu Experimenter a dávkového souboru nad experimentem hledání vhodného nastavení parametrů

zkvalitňuje efektivitu učení v procesu klasifikace.

Tento způsob paralelizace je rovněž jako u předchozí varianty označován jako *embarrassingly paralllel*. Rozdíl oproti předchozímu způsobu je ten, že v tomto případě paralelizuje celkový proces klasifikace, nikoliv pouze jeho část.

5.6 Výhody a nevýhody použití platformy Java™

Bavíme-li se o struktuře nástroje Weka, musíme taky zmínit výhody a nevýhody platformy, na které nástroj běží. V kapitole 2.5 uvádíme, že existují nástroje pro strojové učení psané v různých programovacích jazycích. Z toho také plyne široké spektrum použitých platforem. V této kapitole se zabýváme výhodami a nevýhodami, které nám přináší použití platformy Java SE.

Platforma Java SE je specifická pro použití na osobních počítačích - pro desktopové aplikace [2]. Vyvíjená aplikace v Java SE je pak stavěna pro standalone použití. Použití této platformy nese řadu výhod:

[2] pomineme použití frameworků pro portaci na web aplikaci

- aplikace pro svůj běh využívá celý prostor přiděleného JVM [3]
- portabilita - aplikaci napsanou v jazyce Java lze spustit stejným způsobem na UNIX, Windows i OS X (a dalších) systémech
- desktopové aplikace lze spustit vícenásobně na jednom stroji, což do jisté míry zefektivňuje sekvenčně psané aplikace [4]
- možnost spuštění z příkazové řádky, díky kterému dokážeme vzdáleně vyvolat spuštění aplikace (v případě nástroje Weka i konkrétní klasifikační úlohy)
- možnost nastavení JVM při spuštění - 32/64bitový režim, alokace různé velikosti paměti pro různé instance aplikace
- open source, uživatelská základna
- rozhraní JNI pro integraci knihoven psaných v jiných jazycích

Takto volané nativní funkce mají ovšem několik nevýhod:

- postrádají typovou bezpečnost
- ztrácíme nezávislost na platformě
- JNI postrádá prostředky pro automatické uvolňování nepotřebných zdrojů z paměti - o to se musí starat nativní funkce

Další nevýhody platformy Java SE:

- pomalý start aplikace způsobený překladem a interpretací virtuálního stroje
- relativně velká paměťová náročnost u méně náročné aplikace
- nutnost virtuálního stroje

Nástroj Weka dokáže těchto výhod naplno využít. Pokud nás zajímá analýza velkých dat, potom zajisté uvítáme možnost nastavení JVM - převážně tedy nastavení velikosti alokované paměti. Kombinací ostatních výhod také docílíme vzdáleného spuštění více instancí nástroje Weka, díky čemuž můžeme naplno využít potenciál nabízených prostředků. Tyto vlastnosti platformy a nástroje Weka nám také dovolují provádět klasifikační úlohy pomocí nástroje Weka na výpočetních uzlech HPC. Touto možností se zabýváme v kapitole 7.

[3] existuje možnost spustit více aplikací v jednom JVM, ovšem přináší to s sebou řadu rizik a omezení a nejedná se o standard

[4] V experimentu kapitoly 5.5.3 jsme se přesvědčili o tom, že v případě nástroje Weka, vícenásobné spuštění na jednom stroji pracuji efektivněji, než sekvenční přístup

6 Zpracování velkých datasetů v nástroji Weka

V této kapitole jsme se zaměřili na výkonnost nástroje Weka z hlediska využití OP počítače při práci nad většími datasety. Také jsme hledali hranici velikosti datasetu, který již nástroj Weka nedokáže zpracovat. V tomto experimentu byly porovnány všechny režimy, které nástroj Weka nabízí.

6.1 Popis experimentu

Jelikož nástroj Weka při normálním zpracování načítá vstupní dataset do OP počítače, existuje zde určitá hranice, za kterou již nástroj nedokáže dataset zpracovat. Cílem tohoto experimentu bylo ukázat, kde se tyto hranice nachází při používání nástroje Weka a jak tyto hranice posunout.

Pro demonstraci byly použity datasety ze stejné problematiky, jako v kapitole 4. Byly použity celkem 3 datasety, které obsahují informace o síťovém provozu. Test byl proveden na osobním počítači (konfigurace je uvedena v kapitole 4.1.2). Jelikož je nástroj Weka napsán v jazyce Java, který umožňuje spouštět aplikace nad libovolnou architekturou, zvolili jsme 64bit verzi pro provedení experimentu. Důvodem byla také možnost alokace většího množství paměti (32bit verze je omezena na 2GB).

Pro zpracování byly použity celkem čtyři metody:

- zpracování pomocí režimu Explorer
- zpracování pomocí režimu KFI v režimu práce s celým datasetem
- zpracování pomocí režimu KFI v režimu inkrementálního učení
- zpracování pomocí režimu CLI

Weka Explorer je základní přístup ke zpracování datasetů v nástroji Weka. Obsahuje grafické rozhraní a umožňuje dataset načíst, provádět nad ním různé operace a až následně jej klasifikovat. Proto je pro nás důležité zjistit, jak je prostorově náročná práce v tomto režimu.

Režim KFI nám umožňuje vytvořit klasifikační proces obsahující veškeré kroky, které v analýze datasetu potřebujeme. V tomto režimu jsme zvolili dva způsoby zpracování - práci s celým datasetem a práci v režimu inkrementálního učení. Tyto metody se liší ve zvoleném algoritmu. Zvolili jsme oba přístupy z toho důvodu, že zpracování inkrementálním algoritmem je oproti zpracování standardním algoritmem značně odlišné.

Režim CLI je velice důležitý pro vzdálené vyvolání analyzační úlohy, jelikož nepotřebuje grafické rozhraní a lze jej spustit z příkazové řádky.

6.2 Popis dat

První dataset (*kddcup.data*) má velikost 693 MB a obsahuje bezmála 5 miliónů záznamů. Každý záznam disponuje 41 parametry síťového provozu a posledním parametrem, typem útoku. Typu

Tabulka 9: Využití OP a rychlost klasifikace při analýze datasetů v režimu Explorer

Dataset:	kddcup.data	kddcup.unlabeled	kddcup.data_10_pct
Velikost:	693 MB	409 MB	70 MB
OP před načtením:	145 MB	145 MB	145 MB
OP po načtení:	3 612 MB	2 821 MB	701 MB
OP po klasifikaci:	- (OOM)	4 187 MB	1 577 MB
Doba klasifikace [mm:ss]:	-	0:39	1:33
Relativní výkonnost:	**20%/-**	**15,28%/10,11%**	**12,6%/4,88%**

Tabulka 10: Využití OP a rychlost klasifikace při analýze v režimu KFI v režimu práce s celým datasetem

Dataset:	kddcup.data	kddcup.unlabeled	kddcup.data_10_pct
Velikost:	693 MB	409 MB	70 MB
OP před načtením:	345 MB	345 MB	345 MB
OP po načtení:	3 339 MB	2 600 MB	664 MB
OP po klasifikaci:	- (OOM)	3 600 MB	1 160 MB
Doba klasifikace [mm:ss]:	-	0:48	1:00
Relativní výkonnost:	**23,2%/-**	**18,13%/12,56%**	**21,9%/8,59%**

útoků bylo celkem 24 a byly zařazeny do 4 hlavních skupin útoků (+ pátá skupina, která neobsahuje útoky). Pro demonstraci byl na tomhle datasetu použit algoritmus J48 a data byla rozdělena na 66% tréninková a 34% testová. Pro inkrementální učení byl zvolen naivní bayesovský algoritmus v inkrementální podobě.

Druhý dataset (*kddcup.unlabeled*) má velikost 409 MB a obsahuje bezmála 3 milióny záznamů. Rovněž každý záznam disponuje 41 parametry síťového provozu, ovšem tento dataset neobsahuje třídy útoků (jedná se o učení bez učitele). Pro demonstraci byl zvolen algoritmus shlukování k-means s 5ti shluky a data byla rozdělena na 66% tréninková a 34% testová.

Třetí dataset (*kddcup.data_10_pct*) o velikosti 70 MB je podmnožinou prvního zmiňovaného datasetu s bezmála 500 tis. záznamy. Dataset obsahuje stejná data, jako první dataset a rovněž jsme použili stejné algoritmy.

6.3 Výsledky

Ve výsledcích zavádíme pojem **relativní výkonnost**. Jedná se o parametr, který ukazuje poměr mezi velikostí datasetu a alokovaným místem v operační paměti. Takto si můžeme udělat představu o tom, jak velký dataset je schopný nástroj Weka zpracovat. Jedná se o parametr orientační, protože data mohou mít různou povahu.

V tabulce 9 si můžeme všimnout u prvního datasetu problému s klasifikací - výpočet byl pro takto velký dataset tak prostorově náročný, až vedl k nedostatku paměti (OOM) virtuálního

Tabulka 11: Využití OP a rychlost klasifikace při analýze v režimu KFI v režimu inkrementálního učení

Dataset:	kddcup.data	kddcup.unlabeled	kddcup.data_10_pct
Velikost:	693 MB	409 MB	70 MB
OP před načtením:	229 MB	-	229 MB
OP po načtení:	837 MB	-	548 MB
OP po klasifikaci:	548 MB	-	548 MB
Doba klasifikace [mm:ss]:	4:28	-	0:28
Relativní výkonnost:	**114,1%/114,1%**	**-**	**21,9%/21,9%**

Tabulka 12: Využití OP a rychlost klasifikace při analýze v režimu CLI

Dataset:	kddcup.data	kddcup.unlabeled	kddcup.data_10_pct
Velikost:	693 MB	409 MB	70 MB
OP před načtením:	0 MB	0 MB	0 MB
OP po načtení:	- (OOM)	4 516 MB	1 593 MB
OP po klasifikaci:	- (OOM)	4 516 MB	1 593 MB
Doba klasifikace [mm:ss]:	-	2:37	1:16
Relativní výkonnost:	**-/-**	**9,05%/9,05%**	**4,39%/4,39%**

stroje. Relativní výkonnost načtení datasetu byla 20%, což je vzhledem k ostatním testovaným datasetům dobrý výsledek.

Druhý dataset v tabulce 9 však nástroj Weka zpracoval úspěšně s relativní výkonností načtení okolo 15%, resp. 10% při klasifikaci. Jelikož se alokovaná paměť pohybovala na hraně paměti maximálně dostupné, můžeme usuzovat, že se jedná o maximální velikost datasetu, který dokáže Weka načíst. V případě zpracování si nemůžeme být tak jisti, jelikož každá metoda pracuje s daty jinak a tudíž při sofistikovanějším výběru metody můžeme dospět k lepší výkonnosti.

Třetí dataset v tabulce 9 vykazuje relativní výkonnost načtení 12,6%, resp. 4,88% při klasifikaci, což poukazuje na trend, že s klesající velikosti datasetu klesá i relativní výkonnost.

Tabulka 10 ukazuje výsledky při práci v režimu KFI nad celým datasetem. Při poohlédnutí na tabulku 9 můžeme pozorovat podobný trend, jen s nepatrně lepšími výsledky relativní výkonnosti. Tento způsob rovněž nedokázal zpracovat první dataset. Poněkud překvapivě zde panuje velká výkonnost třetího datasetu, kterou přisuzujeme lepší optimalizaci režimu KFI.

V tabulce 11 vidíme výsledky inkrementálního učení v režimu KFI. Zde dosahovala relativní výkonnost u prvního datasetu hodnoty 114,1%, což značně zvyšuje maximální možnou velikost datasetu pro zpracování v nástroji Weka. Druhý dataset nebyl zpracován, jelikož nástroj Weka v základní instalaci neobsahuje inkrementální shlukovací algoritmy, díky kterým by bylo zpracování možné. Třetí dataset vykazuje stejnou relativní výkonnost načítání, jako v metodě načtení celého datasetu. Relativní výkonnost klasifikace byla rovněž větší, ovšem oproti prvnímu datasetu jen pomíjivě. Z toho můžeme usuzovat, že inkrementální učení je efektivnější pro větší

datasety.

Tabulka 12 ukazuje výsledky při práci v režimu CLI. Dataset *kddcup.data* se nepodařilo načíst z důvodu nedostatku paměti. Klasifikace datasetu *kddcup.unlabeled* proběhla za 2 min a 37 sekund s relativní výkonností 9,05%. Třetí dataset vykazuje relativní výkonnost 4,39%, což je nejhorší výsledek ze všech pokusů. Obecně je tato metoda přirovnatelná ke zpracování skrze režim Explorer, ovšem s horší výkonností.

6.4 Vyhodnocení experimentu

Výsledky tohoto experimentu poukazují na trend, že zpracování pomocí režimu KFI je výkonnější, než-li zpracování pomocí režimu Explorer. Celkově se ale výsledky pohybovaly ve velice malém měřítku oproti inkrementálnímu učení. Tam se hranice velikosti datasetu pohybují v násobcích oproti ostatním variantám práce s celým datasetem. V zpracování skrze CLI jsme se dostali k nejhorším výsledkům. Ovšem nespornou výhodou tohoto zpracování je vyvolání skrze příkazovou řádku.

Za zmínku také stojí fakt, že zpracování v 64bitové verzi Javy je paměťově náročnější, než zpracování v 32bitové verzi. Pravděpodobně bychom tedy v 32bitové verzi dosáhli lepších výsledků u využití OP. Avšak hranice velikosti datasetu, který by 32bitová verze dokázala zpracovat, by byla menší. Proto jsme od této možnosti upustili a raději zvolili 64bitovou verzi.

Obecně můžeme tvrdit, že ke zpracování velkého datasetu v nástroji Weka není vhodné použít osobní počítač. Ten je vhodný pro analýzu menších a středních datasetů (cca do 400MB - záleží na zvoleném algoritmu, zvolené metrice evaluace klasifikace a také na systémových prostředcích). Při zpracování většího datasetu bychom již měli uvažovat nad jinými postupy. Nabízí se použití balíčku *distributedWekaHadoop*. Další možností je využití výpočetních uzlů HPC clusteru, které disponují mnohonásobně většími prostředky. Právě režim CLI umožňuje vytváření úloh ve formátu, jaký vyžadují výpočetní uzly HPC clusteru.

7 Weka v oblasti HPC

V předchozích kapitolách jsme se zabývali analýzou nástroje Weka na osobním počítači. Analyzovali jsme také možnosti paralelizace tohoto nástroje, efektivitu práce s pamětí a s procesorem. Také byly ukázány možnosti spuštění skrze příkazovou řádku a ostatní režimy nástroje Weka. Rovněž jsme ukázali, jaké jsou možnosti rozšíření nástroje a jak pracuje nástroj s velkými datasety. Veškeré tyto aktivity byly předpoklady pro to, abychom mohli nástroj Weka efektivně nasadit na výpočetní uzly clusteru.

V této kapitole je stručně představen cluster, na kterém je nástroj spouštěn. Dále je uveden postup, jak lze nástroj Weka na clusteru spustit a jaké jsou jeho možnosti spuštění. Také jsme se zabývali možností paralelního zpracování klasifikační úlohy na clusteru. V neposlední řadě budou uvedeny výsledky analýzy datasetů z oblasti detekce průniků, kde jsme se zaměřili na efektivitu zpracování velkých datasetů pomocí nástroje Weka na clusteru. Na závěr byly ze získaných výsledků vytvořeny předpoklady pro efektivní práci s nástrojem Weka na clusteru.

7.1 Salomon

Salomon je superpočítačový cluster, který je umístěn v prostorách IT4Innovations. V červnu roku 2015 se jednalo o 40. nejvýkonnější cluster v žebříčku TOP500 [34] a zároveň nejvýkonnější cluster s koprocesory Intel Xeon Phi v Evropě.

Salomon obsahuje celkem 1008 výpočetních uzlů z čehož 576 jsou běžné výpočetní uzly a 432 jsou akcelerované výpočetní uzly, dosahující teoretického výkonu 2011 Tflop/s [35]. Disková kapacita je 2 PB a 3 PB záložní diskové kapacity. Každý výpočetní uzel je osázen dvěma dvanáctijádrovými procesory Intel Xeon E5-2680v3, 2.5GHz, s architekturou x86-64 a RAM 128GB, 5,3GB na jádro, DDR4@2133 MHz. Jednotlivé uzly jsou propojeny InfiniBand a Ethernet sítěmi. Všechny uzly sdílí paměťový prostor 0,5 PB pro domovské adresáře uživatelů a 1,69 PB pro projektová data. Přístup uživatelů je zajištěn čtyřmi login uzly. Operační systém používaný na clusteru je CentOS 6.6 Linux.

Na login uzlech mohou uživatelé spouštět jednotlivé skripty a alokovat si potřebný procesorový čas skrze následující typy front:

- qexp - expresní fronta
- qprod - produkční fronta
- qlong - fronta pro dlouhé úkoly
- qmpp - fronta pro masivní paralelní úkoly
- qfat - fronta pro přístup na stroj SMP UV2000
- qfree - free fronta

```
[stepamic@login2.salomon ~]$ module load Java
[stepamic@login2.salomon ~]$ java -version
java version "1.8.0_51"
Java(TM) SE Runtime Environment (build 1.8.0_51-b16)
Java HotSpot(TM) 64-Bit Server VM (build 25.51-b03, mixed mode)
```

Obrázek 23: Načtení modulu Java a zjištění její verze na login uzlu clusteru Salomon

Jednotlivé fronty jsou určeny pro různé typy úloh - detailnější popis nalezneme v dokumentaci [36]. Námi zvolená fronta *qprod* se vyznačuje maximální dobou běhu 48 hodin. Fronta *qfree* nabízí maximální dobu běhu jedné úlohy 12 hodin.

Každý uzel nabízí možnost načtení modulů. Pro naši práci využijeme moduly *Java* a *Python*. Příkaz pro načtení a pro zjištění aktuální verze modulu Java je na obrázku 23.

7.2 Spuštění nástroje Weka

Pokud chceme provést úlohu klasifikace na výpočetním uzlu clusteru, prvním krokem je přihlášení na login server. Následující příkazy demonstrují postup pro linuxové distribuce. Příkazem

```
$ ssh -X -i id_rsa.sdx.txt stepamic@salomon.it4i.cz
```

kde `id_rsa.sdx.txt` značí umístění privátního RSA klíče a `stepamic` přihlašovácí jméno[5], se přihlásíme na login uzel. Přepínač `-X` umožní budoucí spuštění grafického rozhraní. V ukázce ovšem pracujeme v prostředí bashe. Pro analýzu pomocí nástroje Weka na clusteru je zapotřebí zkopírovat spustitelný archiv do sdíleného úložiště, do kterého máme ze všech uzlů přístup. Ke kopírování souborů jsme použili program WinSCP.

Jakmile máme soubory nástroje Weka ve sdíleném adresáři, můžeme odzkoušet spuštění nástroje Weka. Nejprve jsme si načetli modul Java:

```
$ module load Java
```

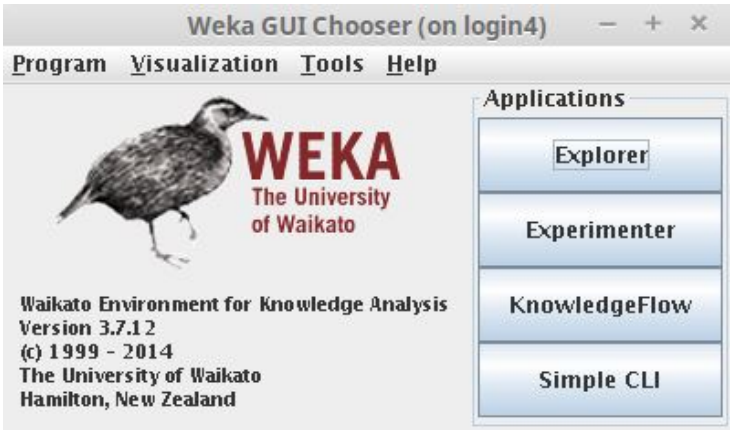
Poté jsme spustili *jar* nástroje Weka skrze příkazovou řádku:

```
$ java -jar -d64 weka/weka-3-7-12/weka.jar
```

Po spuštění se nám zobrazila úvodní obrazovka nástroje Weka (obrázek 24). Zde je nutné zmínit zásadní podmínku - celý běh musí být spuštěn skrze příkazovou řádku. Jediným způsobem, jak alokovat výpočetní čas, prostředky a zajistit spouštění úloh, je zasílání úloh do fronty skrze login uzly. Následujícím příkazem jsme vytvořili požadavek na výpočet naši úlohy:

```
$ qsub -A [projectID] -q [queue] -l select=[x]:ncpus=[y],walltime=[[hh:]mm:]ss
    [.ms] [jobscript]
```

[5] předpokládáme, že uživatel má zřízený přístup na cluster



Obrázek 24: Úvodní obrazovka nástroje Weka spuštěného na login uzlu clusteru

```
[stepamic@login4.salomon ~]$ qsub -A IT4I-7-2 -q qfree -l select=1:ncpus=24,walltime=00:05:00 ./weka-test-job
527066.isrv5
```

Obrázek 25: Navrácené *jobid* pro zadanou úlohu

Požadavek na výpočet úlohy definované v souboru `weka-test-job` přiřazené na projekt `OPEN-0-0` ve frontě `qprod` na 1 uzlu s využitím 24 jader a dedikovaným časem 2 hodiny bude vypadat takto:

```
$ qsub -A OPEN-0-0 -q qprod -l select=1:ncpus=24,walltime=02:00:00 ./weka-test-
    job
```

Obsah souboru `weka-test-job` s definovaním klasifikační úlohy můžeme vidět na výpisu 5.

```
module load Java
cd /home/stepamic/weka/weka-3-7-12
java -cp weka.jar weka.classifiers.trees.RandomForest -num-slots 500 -t data/
    iris.arff -x 10 > HPCoutput.txt
```

Výpis 5: Obsah souboru weka-test-job pro spuštění klasifikační úlohy na výpočetním uzlu HPC

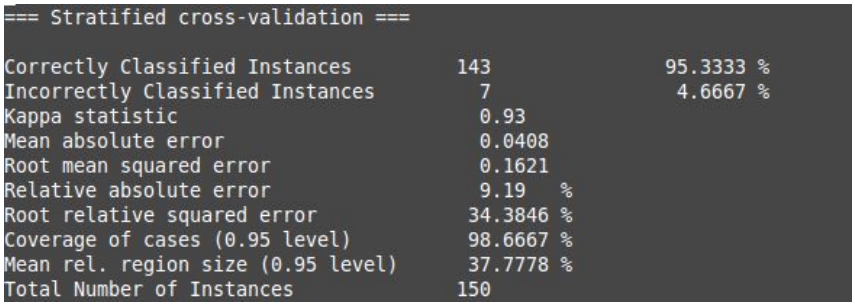
Po zaslání úlohy do fronty nám systém přidělil číslo úlohy (viz. obrázek 25). Aktuální stav úloh vyvoláme následujícím příkazem:

```
$ qstat -u stepamic
```

Kde `stepamic` je uživatel, jehož úlohy chceme zobrazit. Výstupem je seznam aktuálně běžících úloh, znázorněných na obrázku 26. Statistiky běhu úlohy můžeme díky vrácenému *jobid* sledovat

```
isrv5:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
527066.isrv5    stepamic qfree    weka-test-  32694   1  24    --  00:05 R 00:00
```

Obrázek 26: Aktuální fronta úloh uživatele stepamic



Obrázek 27: Část výsledku klasifikace na výpočetním uzlu clusteru - statistika stratifikované křížové validace

po přihlášení na webu [37], kde můžeme mj. sledovat i využití systémových prostředků. Výsledkem je pak provedená klasifikace uložená do výstupního souboru `HPCOutput.txt`. Část obsahu zabývající se kvalitou naučeného algoritmu můžeme vidět na obrázku 27.

Tímhle způsobem jsme provedli klasifikaci pomocí nástroje Weka na výpočetním uzlu clusteru Salomon. Takto provedenou klasifikaci ovšem můžeme jen těžko parametrizovat, jelikož zasíláme stejnou úlohu na každý uzel. Z tohoto důvodu popisujeme možnosti parametrizace a paralelizace v následující kapitole.

7.3 Paralelní zpracování

Pro potřeby parametrizace úlohy na clusteru jsme využili skriptu [38] napsaném v jazyce python. Příklad takové parametrizované úlohy vidíme na výpisu 6.

```
import mpi4py.MPI as MPI
from subprocess import check_call

# Determine process ID, ranks are counter from zero
rank = MPI.COMM_WORLD.rank

command = "java -cp /home/stepamic/weka/weka-3-7-12/weka.jar weka.classifiers.trees.RandomForest -num-slots
    {} -t /home/stepamic/weka/weka-3-7-12/data/iris.arff -x 10 > /home/stepamic/multinode_example/
    HPCoutput_for_{}.txt".format((rank+1) * 200, (rank+1) * 200)

print "On process {} executing command: {}".format(rank, command)
check_call(command, shell=True)
```

Výpis 6: Python skript pro parametrizaci úlohy klasifikace iris datasetu

Tato úloha vygeneruje počet procesů podle zadaného parametru. Pro práci na batch systému je nutné ještě zabalit tento skript spouštěcím shell skriptem (výpis 7).

```
#PBS -l select=2:ncpus=24,walltime=00:30:00

#PBS -q qprod

#PBS -A IT4I-7-2


module load Python/2.7.9-intel-2015b

module load Java

cd /home/stepamic/multinode_example

mpirun -np 2 python multinode.py
```

Výpis 7: Shell skript pro spuštění parametrizovaných úloh klasifikace iris datasetu

Úlohu pošleme do fronty příkazem:

```
$ qsub multinode.sh
```

Výsledkem našeho konkrétního příkladu jsou 2 procesy klasifikace, kde každý je spuštěn na samostatném uzlu a každý s jinými parametry. Výstupy jsou separátně uloženy do složky, ze které jsme skript na login uzlu spustili.

Čeho jsme dosáhli, je popisovaný způsob paralelizace - **paralelizace nástroje** (viz. kapitola 5.5.3). Nástroj Weka je spuštěn na 2 uzlech, kde každá klasifikační úloha má k dispozici 24 jader. Proto jsme zvolili algoritmus Random Forest, kterému jsme schopni nastavit počet vláken pro spuštění (v našem případě 200 a 400). Reálná úloha by pak spočívala v tom, že hledáme optimální počet vláken pro zpracování úlohy na dostupném hardware.

Adekvátně k parametrizaci počtu vláken můžeme přistupovat k paralelizaci různých parametrů algoritmů strojového učení dostupných v nástroji Weka. Toto řešení je ovšem analogické k předchozí úloze, proto jej dále nerozebíráme.

Nástroj Weka není primárně určen pro zpracování na HPC - z toho důvodu chybí nástroji Weka algoritmy, které by dokázaly pro proces učení využít paralelně více, než jeden výpočetní uzel. Proto jsme pro způsob paralelizace formou **paralelizace algoritmů** uvažovali využití jediného výpočetního uzlu (konfigurace výpočetního uzlu je uvedena v kapitole 7.1). Implementace tohoto způsobu paralelizace je podobná, jako na osobním počítači - s tím rozdílem, že je zde kladen větší důraz na paralelizaci (optimalizace počtu vláken).

Poslední způsob paralelizace (**paralelizace procesu klasifikace** - viz. kapitola 5.5.2) není na HPC jednoduchý provést. Nástroje *Weka-Parallel* a režim Experimenter, které se tímto způsobem paralelizace zabývají, nachází své využití spíše v distribuovaných sítích. Praktické využití tohoto způsobu paralelizace je nad rámec této práce.

7.4 Analýza datasetů z oblasti detekce průniků

Tato kapitola se zabývá problematikou, kterou jsme načali v kapitole 4, kde jsme analyzovali dataset z oblasti detekce průniků. Analyzovaný dataset byl malý a proto nebyl problém jej analyzovat na osobním počítači. Nyní byl ovšem analyzován větší dataset, které již na osobním počítači analyzovat nelze (a pokud ano, tak jen částečně - viz. kapitola 6). V experimentu jsme rovněž sledovali využití systémových prostředků. Pro zjednodušení jsme použili statistiky z plánovače PBS [27].

V experimentech prováděných Abrahamem[28] a Valečkem [29] se pracovalo s datasety, které byly podobné těm, použitým v kapitole 4. Oba experimenty ale vycházely ze stejného vstupního datasetu *kddcup.data_10_percent*, který je 10% částí datasetu *kddcup.data.all*. V textu dále referujeme tyto datasety jako menší (*kddcup.data_10_percent*) a větší (*kddcup.data.all*). V této kapitole se zabýváme oběma těmito datasety a analyzujeme je jako celek - nezkoumali jsme úspěšnosti jednotlivých tříd, nýbrž hlavním ukazatelem úspěšnosti klasifikace byla celková úspěšnost klasifikace, evaluovaná pomocí stratifikované 10ti-skupinové křížové validace. Rovněž jsme se v experimentu neomezovali na již použité algoritmy. Analyzovali jsme i algoritmy, které svou implementací mohou vykazovat větší výkonnost při použití na clusteru. V neposlední řadě byly porovnány celkové úspěšnosti klasifikací na obou datasetech. Jednotlivé algoritmy, jež jsme k učení použili, jsme seskupili do skupin. Seskupovali jsme je podle dominantní vlastnosti, kterou se algoritmus při použití na clusteru vyznačoval. V experimentu jsme rovněž sledovali rozdíly v úspěšnosti klasifikací obou datasetů. V experimentu jsme se také zaměřili na dostupné inkrementální algoritmy v nástroji Weka a jejich porovnání se standardními z hlediska výkonnosti v prostředí HPC.

7.4.1 Rozhodovací stromy

Z řad algoritmů rozhodovacích stromů jsme vybrali algoritmy J48 a Random Forest. Parametry algoritmu J48 byly ponechány v původním nastavení. U algoritmu Random Forest jsme zvolili následující nastavení:

1. učení na menším datasetu v původním nastavení
2. učení na menším datasetu s počtem slotů pro zpracování nastaveným na 1
3. učení na menším datasetu s počtem slotů pro zpracování nastaveným na 500
4. učení na větším datasetu s počtem slotů pro zpracování nastaveným na 1000
5. učení na větším datasetu s počtem slotů pro zpracování nastaveným na 1000 a nastavenou agresivní taktikou garbage collectoru s alokovaným prostorem 120 GB

Tabulka 13 ukazuje souhrné statistiky klasifikačních úloh nad oběma datasety pomocí algoritmu J48. Ten je ve své implementaci v nástroji Weka neparalelizovaný, vyznačoval se tudíž malým

Tabulka 13: Statistika klasifikace datasetů z oblasti detekce průniků pomocí algoritmu J48 na clusteru

Dataset:	kddcup.data_10_percent	kddcup.data.all
Alokovaných jader:	24	24
Čas klasifikace:	00:14:18	05:45:16
Alokovaná paměť:	3,4 GB	14,7 GB
Využití CPU:	4,42 %	5,96 %
Úspěšnost klasifikace:	99.968 %	99.994 %

Tabulka 14: Statistika klasifikace datasetů z oblasti detekce průniků pomocí algoritmu Random Forest na clusteru

Nastavení:	1	2	3	4	5
Alokovaných jader:	24	24	24	24	24
Čas klasifikace:	01:05:51	01:09:50	00:05:56	02:32:49	03:17:14
Alokovaná paměť:	2,5 GB	2,57 GB	12,1 GB	32,9 GB	125,5 GB
Využití CPU:	4,71 %	4,63 %	75,3 %	72,17%	68,58%
Úspěšnost klasifikace:	99,983 %	99.983 %	99,983 %	99,937%	99,997%

využitím výpočetních prostředků (100% = maximální využití všech 24 CPU). Můžeme pozorovat, že úspěšnost klasifikace byla u datasetu *kddcup.data.all* nepatrně lepší. Je to dané větším množstvím dat, na kterém se mohl model učit.

Tabulka 14 ukazuje statistiky učení modelu algoritmu Random Forest. V nastavení 1 proběhlo učení za znatelně delší dobu oproti algoritmu J48. Alokovaná paměť byla menší, ovšem pouze nepatrně. Z hlediska využití CPU a úspěšnosti klasifikace si byly algoritmy velice podobné. Nastavení 2 se od nastavení 1 liší jen nepatrně. Z toho můžeme soudit, že původní nastavení má nastaven počet slotů na 1. Nastavení 3 vykazovalo mnohem efektivnější klasifikaci z hlediska doby učení, které bylo o více než 90% kratší oproti předchozím variantám. Rovněž tak využití CPU bylo poměrově o více než 90% větší při zachování stejné úspěšnosti klasifikace. Vhodným nastavením počtu slotů jsme docílili slušné úrovně paralelizace klasifikace. Nastavení 4 algoritmu Random Forest se učilo na větším datasetu o více než polovinu kratší dobu, než se učil algoritmus J48. Využití paměti dosáhlo hodnoty 32 GB, což je dvojnásobek paměti využité při učení pomocí algoritmu J48. Využití CPU bylo podobné jako v třetím nastavení. Úspěšnost klasifikace se pohybovala na průměru úspěšnosti celkové klasifikace pomocí rozhodovacích stromů. V nastavení 4 jsme dosáhli maximální doporučené hodnoty využité OP pro garbage collector. Pro účely experimentu jsme se rozhodli zvýšit tuto hodnotu na neoptimalizovanou hodnotu. Nastavení 5 s pozměněným chováním GC ukazuje, že i když je využití OP bezmála 4x větší, než u předchozí varianty, doba učení se nezkrátila - naopak, prodloužila se zhruba o 30%. Neplatí tedy stanovisko, že s větší dostupnou pamětí roste efektivita klasifikace (z hlediska využití CPU a času klasifikace). Tento fakt nemusí být způsoben nevhodnou implementací algoritmu nebo

Tabulka 15: Statistika klasifikace datasetů z oblasti detekce průniků pomocí algoritmu PART na clusteru

Dataset:	kddcup.data_10_percent	kddcup.data.all
Alokovaných jader:	24	24
Čas klasifikace:	00:17:46	07:53:55
Alokovaná paměť:	2,9 GB	14,7 GB
Využití CPU:	4,16 %	6,88 %
Úspěšnost klasifikace:	99,976 %	99,995 %

nastavením GC, nýbrž také infrastrukturou zvoleného clusteru. Na výpočetních uzlech clusteru Salomon jsou 2 procesory, kde každý z nich má svou paměťovou banku a pokud se algoritmus nenapíše správně v OpenMPI nebo MPI, může docházet k tomu, že procesor P1 s paměťovou bankou PB1 přistupuje do paměťové banky PB2 procesoru P2, do které je přístup pomalejší. Díky tomu může dojít k tomu, že se učení modelu nezrychlí. Z našich experimentů plyne poučení, že je lepší ponechat GC v původním nastavení.

7.4.2 Asociační pravidla

Ze skupiny algoritmů asociačních pravidel jsme vybrali algoritmus PART. Algoritmus byl pro obě úlohy ponechán v původním nastavení.

V tabulce 15 můžeme pozorovat souhrnné statistiky pro algoritmus PART. Obě úlohy nedostatečně využívaly výpočetní prostředky, což je zapříčiněné chybějící paralelizací algoritmu PART. Jelikož jsou si asociační pravidla a rozhodovací stromy velice podobné, můžeme je porovnat i mezi sebou.

Z hlediska času klasifikace si vedl algoritmus PART oproti J48 hůř zhruba o 20-30%. Oproti algoritmu Random Forest byla doba klasifikace zhruba 3x delší. Algoritmus PART ovšem dosahoval větší průměrné úspěšnosti klasifikace v obou úlohách. Z pohledu využití CPU si byly oba algoritmy rovnocenné.

7.4.3 Neuronové sítě

Z oblasti neuronových sítí jsme zvolili algoritmus Multilayer perceptron. Zvolili jsme následující nastavení algoritmu:

1. učení na menším datasetu v původním nastavení
2. učení na menším datasetu s rozděleným datasetem na 66% tréninková a 34% testovací data, počtem epoch 200 a momentum 0,5
3. učení na větším datasetu s rozděleným datasetem na 66% tréninková a 34% testovací data, počtem epoch 100 a momentum 0,5

Tabulka 16: Statistika klasifikace datasetů z oblasti detekce průniků pomocí algoritmu MP na clusteru

Nastavení:	1	2	3
Alokovaných jader:	24	24	24
Čas klasifikace:	06:00:00 *	02:08:15	01:01:58
Alokovaná paměť:	11,2 GB	10,6 GB	16,5 GB
Využití CPU:	4,08 %	4,17 %	5,46 %
Úspěšnost klasifikace:	-	99,354 %	99,914 %

Tabulka 17: Statistika klasifikace datasetů z oblasti detekce průniků pomocí algoritmu SVM na clusteru

Dataset:	kddcup.data_10_percent	kddcup.data.all
Alokovaných jader:	24	24
Čas klasifikace:	06:00:00 *	15:00:00 *
Alokovaná paměť:	28 GB	31,3 GB
Využití CPU:	4,71 %	9,5 %
Úspěšnost klasifikace:	-	-

Jednotlivé úpravy původního nastavení jsme prováděli z důvodu velké časové náročnosti učení.

Tabulka 16 vykazuje statistiky učení modelu pro algoritmus MP. Kvůli výpočetní náročnosti křížové validace nestihlo proběhnout učení modelu ve stanoveném čase 6 hodin pro nastavení 1. Nastavení 2 zvládlo učení modelu poměrně rychle, ovšem díky rozdělení datasetu na tréninkovou a testovací část mohou být výsledky zkresleny. Zvláště v případě, kdy některá cílová třída obsahuje pouze málo prvků a nachází se v určité části datasetu. U třetího nastavení proběhlo učení nejrychleji, ovšem rovněž za cenu možného zkreslení výsledků. Paměťová náročnost byla oproti algoritmům PART a J48 průměrně 3x větší u menšího datasetu. Zpracování většího datasetu z hlediska využití OP proběhlo podobně, jako v předchozích experimentech.

7.4.4 SVM

Dalším použitým algoritmem byl SVM. Algoritmus byl ponechán v původním nastavení, pouze velikost cache paměti jsme oproti původním 40 MB navýšili na 5 GB.

Výsledky v tabulce 17 ukazují, že čas alokovaný pro učení modelu algoritmu SVM nebyl dostatečný. Z hlediska velmi malého využití CPU a dlouhé doby učení není implementace této metody v nástroji Weka vhodná pro klasifikaci velkých dat. Velikost využité OP se pohybovala na hranici 32 GB. Alokace většího paměťového prostoru skrze úpravu GC by pravděpodobně nepřinesla lepší výsledky (viz výsledky v sekci 7.4.1).

Tabulka 18: Statistika klasifikace datasetů z oblasti detekce průniků pomocí meta algoritmu Bagging na clusteru

Nastavení:	1	2	3	4
Alokovaných jader:	24	24	24	24
Čas klasifikace:	12:00:00 *	12:00:00 *	04:52:39	00:17:34
Alokovaná paměť:	20,2 GB	20,2 GB	16,1 GB	16,2 GB
Využití CPU:	7,33 %	7,88 %	5,67 %	80,8 %
Úspěšnost klasifikace:	-	- %	99,994 %	99,968 %

7.4.5 Meta algoritmy

Ze skupiny meta algoritmů jsme vybrali algoritmus *Bagging*. Pro tento algoritmus jsme zvolili následující nastavení:

1. učení na menším datasetu pomocí algoritmu SVM s počtem paralelních slotů 500 a 1 iterací
2. učení na větším datasetu pomocí algoritmu SVM s počtem paralelních slotů 500 a 1 iterací
3. učení na větším datasetu pomocí algoritmu J48 s počtem paralelních slotů 500 a 1 iterací
4. učení na menším datasetu pomocí algoritmu J48 s počtem paralelních slotů 500 a 24 iteracemi

Tabulka 18 ukazuje výsledky učení meta algoritmu Bagging nad datasety z oblasti detekce průniků. Pro první dvě nastavení nebyl alokovaný čas dostatečný a z toho důvodu nestihlo učení proběhnout. V použití algoritmu Bagging byl algoritmus SVM efektivnější co se týče OP a to zhruba o třetinu v obou případech (oproti nastavení bez algoritmu Bagging). Využití CPU se v prvních dvou nastaveních pohyboval v podobných hodnotách, jako při pouhém použití algoritmu SVM. Učení v nastavení 3 bylo o 15 % rychlejší, než bez použití algoritmu Bagging. Paměťová náročnost a využití CPU byly velice podobné, úspěšnost klasifikace dokonce stejná. Čtvrtá varianta paralelně zpracovala všechny iterace a oproti variantě bez algoritmu Bagging trvala klasifikace o 10 % déle. Využití CPU bylo skoro maximální, úspěšnost klasifikace byla stejná, jako bez použití algoritmu Bagging.

I přes navýšení počtu slotů jsme v prvních třech nastaveních oproti původním variantám algoritmů J48 a SVM nedosáhli lepšího využití CPU. Ve čtvrtém nastavení jsme paralelně spustili 24 iterací a za mírně horší čas jsme obdrželi o poznání přesnější výsledky - jedná se o průměr všech 24 iterací. Zároveň jsme u čtvrté varianty zaznamenali největší využití CPU. Počet paralelních slotů u meta algoritmů tedy slouží k lepšímu využití prostředků pro neparalelní algoritmy.

Tabulka 19: Statistika klasifikace datasetů z oblasti detekce průniků pomocí naivních bayesovských algoritmů na clusteru

Nastavení:	1	2	3
Alokovaných jader:	24	24	24
Čas klasifikace:	00:01:43	00:17:02	00:17:09
Alokovaná paměť:	10,5 GB	13,9 GB	13,6 GB
Využití CPU:	- (neznámé)	11,63 %	10,29 %
Úspěšnost klasifikace:	92,155 %	93,83 %	93,83 %

7.4.6 Bayesovské algoritmy

Baysovské algoritmy jsme testovali v následujících nastaveních:

1. učení na menším datasetu
2. učení na větším datasetu
3. učení na větším datasetu inkrementálním klasifikátorem

V tabulce 19 pozorujeme výsledky učení pro skupinu naivních bayesovských algoritmů. Ve všech variantách jsme zaznamenali několikanásobně rychlejší klasifikaci, než tomu bylo u předchozích algoritmů. Alokovaná paměť se rovněž pohybovala na nízkých úrovních v porovnání s předchozími algoritmy. Úspěšnost klasifikace všech variant ovšem dosahuje nízkých hodnot.

Zajímavé je porovnání druhého a třetího nastavení. V kapitole 6 jsme porovnávali standardní a inkrementální algoritmy z hlediska využití OP a načtení datasetu na osobním počítači. Zde můžeme pozorovat, že doba klasifikace i alokovaná paměť si byly v obou nastaveních velice podobné. Může to být dané obrovskou dostupnou OP uzlu, díky které má JVM k dispozici větší prostor a je benevolentnější při rozdělování paměťového prostoru.

7.4.7 Inkrementální algoritmy

Nástroj Weka obsahuje v základní verzi pouze několik inkrementálních algoritmů, proto jsme se rozhodli otestovat všechny použitelné v naší problematice. Vytvořili jsme následující nastavení:

1. algoritmus *Hoeffding tree*, učení na větším datasetu
2. algoritmus *IBk*, učení na větším datasetu
3. algoritmus *KStar*, učení na větším datasetu
4. algoritmus *LWL* s nastavením algoritmu J48, učení na větším datasetu

Tabulka 20 ukazuje statistiky učení modelu pro různé inkrementální algoritmy. Jediný algoritmus, který dokázal ve stanoveném čase provést učení, byl algoritmus Hoeffding tree. Doba učení

Tabulka 20: Statistika klasifikace datasetů z oblasti detekce průniků pomocí inkrementálních algoritmů na clusteru

Nastavení:	1	2	3	4
Alokovaných jader:	24	24	24	24
Čas klasifikace:	01:01:58	12:00:00 *	12:00:00 *	12:00:00 *
Alokovaná paměť:	16,5 GB	12,6 GB	14,1 GB	18,7 GB
Využití CPU:	5,46 %	4,08 %	4,75 %	6,66 %
Úspěšnost klasifikace:	99,952 %	-	-	-

Tabulka 21: Statistika klasifikace datasetů z oblasti detekce průniků pomocí algoritmu k-means na clusteru

Nastavení:	1	2
Alokovaných jader:	24	24
Čas klasifikace:	00:00:15	00:02:04
Alokovaná paměť:	1,6 GB	14 GB
Využití CPU:	- (neznámé)	22,7 %
Úspěšnost klasifikace:	71,5 %	76,46 %

byla oproti ostatním klasifikačním algoritmům krátká. Z hlediska využití CPU a OP se nejednalo o nejefektivnější algoritmus, ale realtivně velká úspěšnost klasifikace v kombinaci s krátkou dobou učení dělá z tohoto algoritmu (ze skupiny inkrementálních algoritmů nástroje Weka) vhodný algoritmus pro efektivní zpracování datasetů z oblasti detekce průniků. Ostatní varianty inkrementálních algoritmů byly o poznání časově náročnější a učení pomocí těchto algoritmů nestihlo proběhnout v alokovaném čase.

7.4.8 Shlukovací algoritmy

Ze skupiny shlukovacích algoritmů jsme vybrali algoritmus k-means v následujícím nastavení:

1. menší dataset, počet paralelních slotů 24, počet shluků 5
2. větší dataset, počet paralelních slotů 24, počet shluků 5

V tabulce 21 pozorujeme výsledky učení pro algoritmus k-means. Učení proběhlo v obou nastaveních velice rychle a jednalo se o nejrychlejší klasifikaci mezi všemi použitými algoritmy. Ve využití OP byla první varianta rovněž nejlepší v porovnání s ostatními algoritmy, druhá varianta se nacházela v průměru. Využití CPU bylo v obou nastaveních nadprůměrné, ovšem nedosahovalo hodnot nejlepší algoritmů. Z pohledu úspěšnosti klasifikace se jednalo o nejméně úspěšné algoritmy.

7.4.9 Vyhodnocení experimentu

Experiment si kladl za cíl provést analýzu datasetů z oblasti detekce průniků v prostředí clusteru. V tomto vyhodnocení se nezabýváme úspěšností celkové klasifikace, nýbrž efektivity zpracování z hlediska využití systémových prostředků a času učení. U algoritmů, kde nástroj Weka zaznamenal při zpracování na osobním počítači nedostatek paměti a klasifikaci tedy nebylo možné provést, tento problém při zpracování na HPC nenastal. Učení na HPC ovšem provázel jiný problém - nedostatečná paralelizace většiny zkoumaných algoritmů.

Algoritmy s **paralelním přístupem** se vyznačovaly vysokým využitím CPU. Do této skupiny řadíme algoritmy Random Forest, k-means a meta algoritmy. Meta algoritmy nezvyšují efektivitu použitých algoritmů, ovšem díky schopnosti spouštět více algoritmů současně dokáží využít efektině prostředky HPC uzlu. Do této kategorie řadíme i bayesovské algoritmy, i když jejich paralelní implementace nebyla tak efektivní. Časy učení algoritmů s paralelním přístupem byly řadově násobně kratší, než u algoritmů se sekvenčním přístupem.

Algoritmy se **sekvenčním přístupem** se vyznačovaly nízkým využitím systémových prostředků. U některých algoritmů (MP, SVM) můžeme pozorovat výrazně delší dobu učení, než u ostatních algoritmů. Proto také algoritmus SVM nestihl naučit model ve stanoveném čase (algoritmu MP se to povedlo za předpokladu snížení počtu epoch a absence křížové validace). Tyto algoritmy také využívaly nejvíce OP. Ostatní sekvenční algoritmy však využívaly méně OP, než tomu bylo u algoritmů s paralelní implementací. Obecně nejsou algoritmy se sekvenčním přístupem vhodné k práci v oblasti HPC z důvodu absence paralelizace a tedy malého využití systémových prostředků. Vhodnější je tyto algoritmy využít pro analýzu malých a středních datasetů.

Inkrementální algoritmy se rovněž potýkaly s nedostatečným využitím CPU. Z testovaných algoritmů dokázal pouze algoritmus Hoeffding Tree naučit model ve stanoveném čase. Jejich paměťová náročnost byla podobná, jako u sekvenčních algoritmů. Výhoda inkrementálního přístupu je v zpracování teoreticky neomezeně velkého datasetu. Zpracování na výpočetním uzlu HPC ovšem není vhodné z důvodu nízkého využití CPU a dlouhé doby učení modelu.

Pokud bychom chtěli dosáhnout efektivního využití dostupných HW prostředků na uzlech clusteru pomocí nástroje Weka, musíme využít jeden z následujících přístupů:

- **Použijeme algoritmus podporující paralelní zpracování.** Takový algoritmus můžeme škálovat podle dostupných prostředků (viz. kapitola 5.5.1).
- **Použijeme meta algoritmus pro zkvalitnění učení sekvenčního algoritmu.** Čas učení nezkrátíme, ovšem vhodným nastavením škálování meta algoritmu zkvalitníme výsledné učení sekvenčního algoritmu.
- **Využijeme paralelizace na úrovni nástroje.** Tímto způsobem jsme schopni spustit několik klasifikací na jediném uzlu zároveň a dosáhnout tedy plné utilizace dostupných prostředků (viz. kapitola 5.5.3).

8 Závěr

Cílem této práce bylo prostudovat možnosti nástroje Weka se zaměřením na koncepci tohoto nástroje a na postupy propojení jednotlivých modulů a toků dat. Dalším cílem bylo zaměřit se na použití nástroje Weka pro zpracování dat velkých objemů nasazením na výpočetní uzly HPC clusteru.

Úvod práce se zabýval uvedením do kontextu strojového učení. Představili jsme si skupiny algoritmů, které se ve strojovém učení a nástroji Weka používají. Následně jsme představili všechny možnosti používání nástroje Weka. Provedli jsme experiment, ve kterém jsme analyzovali datasety z oblasti detekce průniků a na kterém jsme demonstrovali možnosti nástroje Weka. V průběhu experimentu jsme vytyčili slabé místo nástroje Weka - a to načítání datasetu do OP počítače. Tento slabý bod jsme dále analyzovali a hledali jsme řešení, jak jeho dopad na kvalitu učení minimalizovat. V technické analýze nástroje jsme rovněž identifikovali jednotlivé moduly tvořící nástroj Weka a zaměřili jsme se na tok dat mezi nimi. Ze získaných poznatků jsme hledali způsoby, jak proces klasifikace paralelizovat. Objevili jsme celkem 3 způsoby paralelizace. Následně jsme provedli analýzu na clusteru Salomon, kde jsme provedli analýzu dat velkých objemů z oblasti detekce průniků pomocí různých algoritmů, které nástroj Weka nabízí. Možnosti paralelizace jsme na clusteru dále zkoumali a z nabytých zkušeností jsme vytvořili základní principy, díky kterým dokážeme nástroj Weka efektivně používat na clusteru Salomon.

Tato práce přináší poznatky o možnostech využití nástroje Weka a to zejména v oblasti HPC, pro kterou není nástroj primárně určen. Také jsme přinesli poznatky a výsledky analýzy velkých datasetů z problematiky detekce průniků a to díky možnosti analýzy na stroji s velkou pamětí. Provedený rozbor nástroje přináší nové poznatky o tom, jak spolu jednotlivé moduly komunikují a jaké jsou možnosti paralelizace v nástroji Weka. Pro budoucí výzkum je možné vytyčené způsoby paralelizace dále rozebírat a provést jejich implementaci tak, aby byly přizpůsobené pro použití na HPC.

Tato práce může sloužit nejen pokročilým uživatelům nástroje Weka. Myšlenky, s kterými jsme si v práci zahrávali, jsou podobné pro většinu nástrojů analýzy dat. Ta nejzásadnější je následující - můžeme mít sebevíce optimalizovaný algoritmus, ale pokud nemáme vhodný nástroj, který algoritmus efektivně interpretuje, pak je i výsledné učení neefektivní. Toto samozřejmě platí pro případy, kdy nástroj k analýze potřebujeme.

Michal Štěpán

Literatura

[1] SOUČEK, Martin. *UK UISK modul č. 3 – Informační věda*, v rámci projektu Studium informační vědy a komunikačního managementu v evropském kontextu [online]. Informační věda, 2009. [cit. 2016-04-05]. Dostupné z: http://www.informacniveda.cz/article.do?articleId=1130

[2] WITTEN, Ian H. – FRANK, Eibe. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[3] BISHOP, Christopher M. *Pattern Recognition and Machine Learning*, 2006.

[4] RUSSELL, Stuart J – Peter NORVIG. *Artificial intelligence: a modern approach*. 2nd ed. Upper Saddle River, N.J.: Prentice Hall/Pearson Education, c2003. ISBN 0137903952.

[5] MAGERMAN, David M. *Statistical decision-tree models for parsing*. In: Proceedings of the 33rd annual meeting on Association for Computational Linguistics. Association for Computational Linguistics, 1995. p. 276-283.

[6] TIANYU, Zheng. *Decision Tree and Entropy algorithm* [online]. ZhengTianyu's blog, 2013. [cit. 2016-04-02]. Dostupné z: https://zhengtianyu.wordpress.com/2013/12/13/decision-trees-and-entropy-algorithm/

[7] MARSLAND, Stephen. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[8] LEWICKI, Michael S. *Artificial Intelligence: Representation and Problem Solving: Decision Trees 2* [online]. Carneige Mellon, 2007. [cit. 2016-04-02]. Dostupné z: http://www.cs.cmu.edu/afs/cs/academic/class/15381-s07/www/slides/041207decisionTrees2.pdf

[9] PEDDABACHIGARI, Sandhya – ABRAHAM, Ajith – THOMAS, Johnson. *Intrusion detection systems using decision trees and support vector machines*. International Journal of Applied Science and Computations, USA, 2004, 11.3: 118-134.

[10] PAGALLO, Giulia – HAUSSLER, David. *Boolean feature discovery in empirical learning*. Machine learning, 1990, 5.1: 71-99.

[11] ZAKI, Mohammed J. – MEIRA JR, Wagner. *Data mining and analysis: fundamental concepts and algorithms*. 2014.

[12] BURGES, Christopher JC. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 1998, 2.2: 121-167.

[13] VONDRÁK, Ivo. *Neuronové sítě* [online]. Vysoká škola báňská - Technická univerzita Ostrava. Fakulta elektrotechniky a informatiky. Katedra informatiky, 2009. [cit. 2016-04-14]. Dostupné z: http://vondrak.cs.vsb.cz/download/Neuronove_site.pdf

[14] RIEDMILLER, Martin. *Machine Learning: Multi Layer Perceptrons* [online]. Albert-Ludwigs-University Freiburg, 2013. [cit. 2016-04-02]. Dostupné z: http://ml.informatik.uni-freiburg.de/_media/documents/teaching/ss09/ml/mlps.pdf

[15] HOLČÍK, Jiří – KOMENDA, Martin (eds.) a kol. *Matematická biologie: e-learningová učebnice* [online]. 1. vydání. Brno: Masarykova univerzita, 2015. ISBN 978-80-210-8095-9. [cit. 2016-04-02]. Dostupné z: http://portal.matematickabiologie.cz/res/f/neuronove-site-jednotlivy-neuron.pdf

[16] *Apache Mahout: Scalable machine learning and data mining* [online]. The Apache Software Foundation, ©2016. [cit. 2016-04-05]. Dostupné z: http://mahout.apache.org/

[17] KRIZHEVSKY, Alex. *cuda-convnet - High-performance C++/CUDA implementation of convolutional neural networks* [online]. Google Project Hosting, 2014. [cit. 2016-04-05]. Dostupné z: https://code.google.com/p/cuda-convnet/

[18] ERICSON, Gary. What is Azure Machine Learning Studio? In: *Microsoft Azure* [online]. 2016-03-09. [cit. 2016-04-05]. Dostupné z: https://azure.microsoft.com/cs-cz/documentation/articles/machine-learning-what-is-ml-studio/

[19] *Weka 3 - Data Mining with Open Source Machine Learning Software in Java* [online]. Machine Learning Group at the University of Waikato, 2016. [cit. 2016-04-05]. Dostupné z: http://www.cs.waikato.ac.nz/ml/index.html

[20] HALL, Mark. Weka and Hadoop Part 1. In: *Mark Hall on Data Mining & Weka* [online]. 2013-10-15. [cit. 2016-04-02]. Dostupné z: http://markahall.blogspot.cz/2013/10/weka-and-hadoop-part-1.html

[21] LEVASSEUR, Yan. *Genetic Programming Classifier for Weka* [online]. Sourceforge, 2013. [cit. 2016-04-02]. Dostupné z: https://sourceforge.net/projects/wekagp/

[22] DRAGONE, Luigi. *Spectral clusterer for WEKA* [online]. 2002. [cit. 2016-04-05]. Dostupné z: http://www.luigidragone. com/datamining/spectral-clustering.html

[23] CELIS, Sebastian – MUSICANT, David R. *Weka-parallel: machine learning in parallel.* Carleton College, CS TR, 2002.

[24] BROWNLEE, Jason. *How To Choose The Right Test Options When Evaluating Machine Learning Algorithms.* In: Machine learning mastery [online]. 2014. [cit. 2016-04-05]. Dostupné z: http://machinelearningmastery.com/how-to-choose-the-right-test-options-when-evaluating-machine-learning-algorithms/

[25] SCARFONE, Karen – MELL, Peter. *Guide to intrusion detection and prevention systems (idps).* NIST special publication, 2007, 800.2007: 94.

[26] *UCI KDD Archive* [online]. University of California, Irvine, 2005. [cit. 2016-04-02]. Dostupné z: http://kdd.ics.uci.edu/databases/kddcup99/

[27] PBSPro Documentation. *IT4Innovations#Docs* [online]. IT4Innovations - národní superpočítačové centrum, 2016. [cit. 2016-04-17]. Dostupné z: https://docs.it4i.cz/pbspro-documentation

[28] ABRAHAM, Ajith – GROSAN, Crina. *Evolving intrusion detection systems.* In: Genetic systems programming. Springer Berlin Heidelberg, 2006. p. 57-79.

[29] VALEČKO, Miroslav. *Analýza dat pomocí Business Intelligence v SQL Serveru.* Ostrava, 2014. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava. Fakulta elektrotechniky a informatiky. Vedoucí práce Jan MARTINOVIČ.

[30] BOUCKAERT, Remco R., et al. *WEKA manual for version 3-7-12.* 2015.

[31] CHANG, Chih-Chung – LIN, Chih-Jen. *LIBSVM: a library for support vector machines.* ACM Transactions on Intelligent Systems and Technology (TIST), 2011, 2.3: 27.

[32] CELIS, Sebastian – MUSICANT, David. *Weka-Parallel* [online]. Sourceforge, 2013. [cit. 2016-04-02]. Dostupné z: https://sourceforge.net/projects/weka-parallel/

[33] What is a Socket? *Java™Documentation* [online]. Oracle, ©1995-2015. [cit. 2016-04-02]. Dostupné z: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

[34] Top500 List - June 2015. *Top500 Supercomputer Sites* [online]. TOP500.org, ©1993-2015. [cit. 2016-04-02]. Dostupné z: http://www.top500.org/list/2015/06/

[35] Salomon Cluster Documentation: Hardware Overview. *IT4Innovations#Docs* [online]. IT4Innovations - národní superpočítačové centrum, 2016. [cit. 2016-04-02]. Dostupné z: https://docs.it4i.cz/salomon/hardware-overview-1

[36] Salomon Cluster Documentation: Resources Allocation Policy. *IT4Innovations#Docs* [online]. IT4Innovations - národní superpočítačové centrum, 2016. [cit. 2016-04-02]. Dostupné z: https://docs.it4i.cz/salomon/resource-allocation-and-job-execution/resources-allocation-policy

[37] *IT4Innovations Salomon: Cluster usage* [online]. IT4Innovations - národní superpočítačové centrum, 2016. [cit. 2016-04-17]. Dostupné z: https://extranet.it4i.cz/rsweb/salomon

[38] *Running multiple tasks with different parameters on Salomon* [online]. Verif Research Group, Vysoká škola báňská - Technická univerzita Ostrava. Fakulta elektrotechniky a informatiky, 2015. [cit. 2016-04-02]. Dostupné z: http://verif.cs.vsb.cz/sb/docs/simplerun.html

A Obsah přiloženého CD

Přiložené CD obsahuje následující adresáře:

- **Diplomová práce** - text diplomové práce ve formátu PDF
- **HPC** - vytvořené skripty, výsledky klasifikace a statistiky běhu analýzy detekce průniků na výpočetních uzlech HPC
- **Data** - vytvořené datasety pro analýzu problematiky detekce průniků v kapitole 4